Introduction to PHP 5 with SQLite

By

Svein Nordbotten, Professor Emeritus

University of Bergen, Norway

Bergen, September 2010
Contents

About the course. ............................................................................................................................................. 6
Session 1: Static web applications .................................................................................................................... 7
   Basics............................................................................................................................................................ 7
   Web applications ........................................................................................................................................... 7
HTML - Hypertext Markup Language ........................................................................................................... 8
HTML Format .................................................................................................................................................. 9
Texts.............................................................................................................................................................. 9
Links ............................................................................................................................................................... 10
Images ............................................................................................................................................................ 11
Lists ............................................................................................................................................................... 11
Tables ............................................................................................................................................................ 12
Forms ............................................................................................................................................................. 13
Frames........................................................................................................................................................... 14
CSS, JavaScript, and XML............................................................................................................................. 15
Session 2: Dynamic applications in PHP ......................................................................................................... 16
   Dynamic web sites ...................................................................................................................................... 16
   CGI and PERL .......................................................................................................................................... 17
   Applications Program Interfaces ............................................................................................................... 18
   PHP Language......................................................................................................................................... 18
   Approach in this course ............................................................................................................................ 19
   Simple example ......................................................................................................................................... 19
   Guessing example .................................................................................................................................... 21
Session 3: Dynamic application without database .......................................................................................... 24
   Market research....................................................................................................................................... 24
   System design .......................................................................................................................................... 24
   Market research....................................................................................................................................... 26
   Market analysis....................................................................................................................................... 30
Session 4: Introducing the SQLite database .................................................................................................... 33
   Dynamic applications and databases...................................................................................................... 33
   SQLite ......................................................................................................................................................... 33
Creation of a reference database to your personal library .................................................. 33
Menu page .............................................................................................................................. 35
Creating and populating a database ..................................................................................... 36
Listing the content of the database ...................................................................................... 38
Searching the database for a book reference ....................................................................... 39
Updating book references in the database .......................................................................... 42
Deleting rows in the database ............................................................................................... 46
Removing all content in the booktable .................................................................................. 47
Session 5: Application with SQLite database ....................................................................... 49
Opinion polls .......................................................................................................................... 49
Application design ............................................................................................................... 49
Application menu ................................................................................................................. 50
Creating records and a list of panel members ..................................................................... 51
Processing, statistics and rotation ....................................................................................... 55
Session 6: Functions in PHP ................................................................................................. 60
Functions ............................................................................................................................... 60
Authorization and authentication ........................................................................................... 61
  Table existence test ............................................................................................................. 65
  Authorization ....................................................................................................................... 66
  PIN code assignment ......................................................................................................... 66
  Authentication ..................................................................................................................... 67
  Function library .................................................................................................................. 68
Logging ................................................................................................................................... 68
  Logging function ................................................................................................................. 69
  Example environment ......................................................................................................... 69
Session 7: File processing ..................................................................................................... 71
  Maintaining files ................................................................................................................ 71
  Fetching files ..................................................................................................................... 71
  Uploading files ................................................................................................................... 74
Session 8: Object-Orientation in PHP .................................................................................. 79
  Object-oriented interface ................................................................................................... 79
  Object-oriented syntax ...................................................................................................... 79
SQLite OO interface syntax ........................................................................................................ 83
Session 9: Login class ............................................................................................................ 86
Authentication and authorization re-visited ........................................................................ 86
HTML form pages .................................................................................................................. 86
PHP script ............................................................................................................................... 88
LOGIN class ........................................................................................................................... 88
Course framework .................................................................................................................. 92
Infrastructure ........................................................................................................................ 92
Implementation ....................................................................................................................... 92
index.htm ................................................................................................................................ 92
registration.htm ..................................................................................................................... 93
validate.php ........................................................................................................................... 94
frame1.htm ............................................................................................................................ 95
list2.htm ................................................................................................................................ 95
about.php ............................................................................................................................. 96
session1.php .......................................................................................................................... 97
The end .................................................................................................................................... 98
A bibliography for further studies ......................................................................................... 99
Preface

This course was given as an online interactive course for several terms in 2006-2007. The online version included quizzes, live examples and performance test for the students. The present extracts of the course text may still be of some interest.

Svein Nordbotten

Bergen, September 2010
About the course.

Author: Svein Nordbotten.

The number of web applications is increasing rapidly. Still, however, most web sites are static.. i.e. they are created with a fixed content that cannot be adjusted to information provided by the visitor on a returned form or by previously saved information in, for example, a database.

PHP is one of several tools that can be used in creating dynamic web applications. Other alternatives are ASP.NET, JAVA, ColdFusion and iHTML to mention a few. PHP 5 includes both object-orientation as a second interface in addition to the earlier functional interface. This permits creating even more easily re-useable software than previously.

PHP has also included an integrated database system, SQLite, which can serve a number of database needs and makes the installation of a separate database system unnecessary.

The author thanks Tor Kristian Bjelland and Paul Glenn for identifying a number of errors and for many constructive comments to earlier drafts of the course content. Additional thanks to Paul Glenn for his carefully proof reading of all texts.

Bergen, January 2006.
Session 1: Static web applications

Basics

This session is a short introduction to Hyper-Text Mark-up Language (HTML) for those not acquainted with this language, and a fast repetition for those already experts in the language.

Web applications

The topic of this course is the design and implementation of web applications. In this context, a web application is a server-based system that can interact with the user and respond with several interrelated pages for display on the user's computer.

We distinguish between 2 categories of applications, static and dynamic. An application is denoted as static if the pages returned have an invariable content. In a static application, the returned pages cannot be modified according to the individual characteristics or behavior of the user. The user makes a request to a host at which the web server processes the request and returns a web page to be displayed on the user's screen. Note that the web server can retrieve a file stored at the host, for example a .jpg file, and use it for composing the web page. The basic web server cannot, however, store or modify files submitted by the clients.

A dynamic application, on the other hand, can modify its responses by adding to the returned page the name of the user, the number of times this particular user has visited the application web site, her account data, course progress, etc. It requires a special program that is capable of additional processing, for example, to process and save data sent by the user or to return data stored in a database to the user on demand. The main objective of this course is to introduce you to the art of developing dynamic web applications.

The Internet was initiated in the 1970's as a further development of the ARPANET. The World Wide Web, WWW, was developed and introduced in 1989 by Tim Berners-Lee and Robert Cailliau at the European Particle Physics Laboratory (CERN) as an Internet tool for collaborative knowledge-sharing. It became very popular in a short time. Today, the WWW comprises a large number of computers that make files available through the Internet according to the HyperText Transfer Protocol, HTTP. Today, it is estimated that more than 300 M people worldwide are using the web.

The visible content of a web file is called a web document. If a web document is prepared according to the HTTP protocol, it can be transferred from a host computer using appropriate software to a requesting client via the Internet. Most documents are formatted by means of the tag-based language HyperText Markup Language, HTML, which is frequently supplemented with some additional tools. If the requesting client has the necessary browser software installed,
the file received can be displayed and, if desired, a new request can be generated, for example by clicking a link in the displayed document.

A **web site** is usually a set of interrelated web files hosted on a computer running a **web server**. The design and implementation of a web site has several aspects:

- **the topic** of the site,
- **the layout** of the pages sent from the site,
- **the functionality** of the site.

The topics of a web site vary and depend on the owner's interests and mission. In this course, we shall **not** discuss whether certain pages are appropriate for web publication, and which are not. It is easy to find examples of both interesting and less interesting pages.

The layout of pages is a fascinating subject. All kinds of **background colors** and **patterns**, **fonts** of different kinds and sizes, etc., are among the layout factors from which the designer can choose. Some pages have **animation** and/or **sound** embedded, others include programs transferred to and acting on the client computer. The layout of a page is an important subject because it probably has a significant impact on how the receivers will perceive the page. So far, the layout has to a large extent been determined by the latest hypes and layout rules. The heuristic design rules offered have usually been based on personal opinions and **limited empirical facts**. Large scale investigations of people's perception of alternative layouts are needed. However, layout is **not** the main subject of this course.

The subject of this course is the **functionality** required to change the web arena from basically **static** to **dynamic** applications. The required functionality is the web site's ability to react to a visitor's behavior over a shorter or longer time period expressed by a series of requests and responses. This is called dynamic because the web pages that are returned to the client depend on the visitor's previous interaction.

Most web sites are still **static**, i.e. each web page is presented in the same way independent of the client and time. **Dynamic** functionality means that the pages that are returned to the clients can be adjusted to previous input from the individual client at or time. The development of dynamic web sites can be approached in many ways. In this course, we limit our discussion to the **functionality** based on the scripting language **PHP** and on the **PHP Application Engine**. However, before we embark on the dynamic aspects, in this session we shall briefly summarize **HTML**.

**HTML - Hypertext Markup Language**

**HTML** was developed from **SGML**, **Standard Generalized Markup Language**, which was approved in **1986** as a standard for marking up documents so they can be stored and read by computers. **HTML** includes only a **smaller** fraction of the features covered by **SGML** and was intended to be a convenient tool for displaying pages to be served to the users by the **WWW**. The most recent version of **HTML** is **4.01**. An **XML**-based version of **HTML 4.01** is **XHTML 1.0**. In this course we refer to the **HTML 4.01** version. To deliver these the **HTML** pages, web
servers, including the Apache servers, were developed. For the client side, a number of browsers were introduced, of which MS Internet Explorer and Netscape have been the dominant.

The remaining part of this session is a short summary of the most basic parts of HTML needed for this course. For more advanced uses of HTML, readers are referred to more advanced literature.

**HTML Format**

To distinguish between the content of the computer file sent to the browser and the resulting page displayed on the user's screen, in this course we shall refer to the former as an HTML page and the latter as a web display. The HTML language is governed by the use of a set of tags. A tag is a text string surrounded by < and > (e.g. `<center>`). In many cases, the tag string is a single character (`<p>`: start of a new paragraph). Some tags are single such as the tag used for comments (`<!-- Comment -->`) Other tags require a corresponding end tag, which is the tag string preceded by a / (`</center>`: end the centered text). These tags and the included text are called tag blocks. Some tags can be nested. There may for example be several paragraphs within a centered text. Many tags include attributes that can be required or optional (`<font face="New Century" size="2" color="blue">`)  

A complete HTML page consists of several parts. A typical basic structure may look like this (line numbering is included in this and other pages for convenient reference, and should not be included in the page):

1. `<!doctype html public "-//w3w//dtd html 4.0 transitional//en">`
2. `<html>`
3. `<head>`
4. `<title>`
5. `<!-- The title of the document may be typed here --></title>`
6. `</head>`
7. `<body>`
8. `<!-- The specific content of the page is typed in the body-block -->`
9. `</body>`
10. `</html>`

Type this page and save it in your server below the document root with a filename, e.g. `blank.htm`. It can then be called from a client, but since it still does not have any content, it will be displayed as a blank screen by the browser.

Note that this is the complete frame for an HTML page, and it will also usually function with default specifications with only `<html> </html>` surrounding your text.

**Texts**

Let us give the page some content:
The purpose of this session is to introduce the course participants to the basic elements of HTML. It is hoped that the introduction will make it possible for the participants to read the HTML pages used in this course, and use the knowledge for preparing their own simple HTML pages in combination with the PHP scripts.

Good luck!

Greetings from the author.

This page is named text.htm in the example. It illustrates how you can specify headings (standard tags are <h1>, <h2>, and <h3>), color the text (16 different colors are predefined: red, blue, green, blue, etc., and many more are available by code representation), paragraphs (<p>), line shifts (<br>), and center text (<center>).

Links

Hypertext is the trademark of HTML. We can easily develop a page that includes a link (using the <a> and </a> tags) to another document, for example, the page discussed in the section above. The <a> tag requires at least one attribute, href, the value of which is the name of the file enclosed in double quotes to which the link refers.
Several links in sequence can be created to form a menu as in the menu to the HTML example of this session.

**Images**

In the age of multi-media, many HTML pages have illustrations. A possibility to include pictures in the pages is therefore required. We know from regular work with computers that pictures can be saved in a number of different file formats, of which the .gif and the .jpg are used in connection with HTML.

Assume that we have an image of a well-known painting by Edward Munch, The Cry, saved in a file named munch.jpg in the same folder as we use for our HTML pages. We can now write an HTML page that includes this image in the page returned for display.

```html
1. <!doctype html public "-//w3c//dtd html 4.0 transitional//en"> <html> <head>
2. <title>image.htm</title>
3. </head>
4. <body> <center> <h2> <font color="#0000FF"> A Munch picture displayed </font> </h2> <p>You requested a page displaying a picture by Edvard Munch. Here it is: </p> <img src="munch.jpg" width="150" height="200" align="middle"> </center> </body>
5. </html>
```

The tag used is `<img>` which can have several attributes, of which `src`, referring to the file in which the image is stored, is **required**. You can easily scale the picture by changing the `width` and `height` attributes in the image tag. The metric unit used is **pixels**. The position of the picture within the displayed page can be controlled by the `align` attribute with a number of possible **alternative values** (including `left`, `middle`, `right`, `top`, `bottom`). Note that the scaling and positioning attributes are **optional**.

**Lists**

We are used to the ability of modern word processor to prepare **numbered** and **unnumbered lists**. HTML has included this ability by the tag pairs `<ol>` </ol> and `<ul>` </ul>.

The page in this example can serve as an illustration of this capability:

```html
1. <!doctype html public "-//w3c//dtd html 4.0 transitional//en">
2. <html>
3. <head>
4. <title>list.htm</title>
5. </head>
6. <body>
7. <center>
8. <h2> <font color="#0000FF">Menu for the example options</font></h2>
9. <p>This example illustrate the basic features of HTML which are:</p>
10. <ul>
11. <li><a href="blank.htm">Blank page</a></li>
12. <li><a href="text.htm">Text page</a></li>
13. <li><a href="link.htm">Page with link</a></li>
```
The `<li>` and `<li>` tags delimit the individual elements, or lines, in the list. Note that in this page we use the unnumbered `<ul>` tag. By changing the start and end tags to `<ol>` and `</ol>`, the elements would be numbered consecutively from 1 and up.

**Tables**

The table tag, `<table>` is very useful in several ways for presenting one (a list ) and two-dimensional tables with or without borders. When you consider the display of the menu in the previous example, it gives an unordered impression. The use of the table tag with associated tags can make it more orderly. Consider the following page that presents the list as a one-dimensional table:

```
1. <!doctype html public "-//w3c//dtd html 4.0 transitional//en">
2. <html>
3. <head>
4. <title>table.htm<title>
5. <head>
6. <body>
7. <center>
8. <h2><font color="#0000FF">Menu for the example options</font></h2>
9. <p>This example illustrate the basic features of HTML which are:
10. <table>
11. <tr><td><a href="blank.htm">1. Blank page</a></td></tr>
12. <tr><td><a href="text.htm">2. Text page</a></td></tr>
13. <tr><td><a href="link.htm">3. Page with link</a></td></tr>
14. <tr><td><a href="image.htm">4. Page with picture</a></td></tr>
15. <tr><td><a href="table.htm">5. Page with table</a></td></tr>
16. <tr><td><a href="form.htm">6. Form page</a></td></tr>
17. <tr><td><a href="frame.htm">7. numbered Frame page</a></td></tr>
18. </table>
19. </center>
20. </body>
21. </html>
```

In addition to the `<table>` tag, we use the tags `<tr>` and `</tr>` to delimit a table row, and the tags `<td>` and `</td>` to mark an element in the row. In this example there is only one element per row, although there are usaull several. In regular tables there is always one element per column in each row. If the cell is empty it is marked by `<td></td>`.
In regular tables, there is usually also a header row with column names. The column names are marked with the tags `<th>` and `<th>`. Each of the table tags can include one or optional attributes for defining size, alignment, fonts, border, etc., making the tags very flexible and useful.

**Forms**

One of the most important properties of HTML is the `<form>` tag that permits sending data to the server. This tag is the key to combining HTML and the PHP language into a tool for creating dynamic applications. The `<form>` tag makes it possible to create pages for the user with different types of input (radio buttons, check boxes, texts, files, etc) and send the input for further processing by the server according to a specified program, for example a PHP script. Note that HTML itself has no facility for processing data on the server. (There are extensions of HTML that permit limited processing on the server).

We shall see a number of applications in the following sessions based on interaction between HTML and PHP scripts. For illustration of the `<form>` tag in this session, a form will be discussed, and at the accepting server side a very simple PHP script will return a message confirming the submitted information.

The HTML form page looks like this:

```
1. <!--DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2. <html>
3. <head>
4. <title>form.htm</title>
5. </head>
6. <body>
7. <center>
8. <h2><font color="#0000FF">A form for sending a file for processing</font></h2>
9. <p>This form can be used for sending a file for alternative processing, Action A or B, and assumes a processing script at the server. In this example, the only action taken by the server is to return a message acknowledging the received file and message.</p>
10. <form action="acknowledge.php" method="post">
11. <table>
12. <tr><td>Message:</td><td><input name="message" type="text"></td></tr>
13. <tr><td>File:</td><td><input name="testfile" type="file" enctype="multipart/form-data"></td></tr>
14. <tr><td>Action A:</td><td><input name="processing" type="radio" value="A"></td></tr>
15. <tr><td>Action B:</td><td><input name="processing" type="radio" value="B"></td></tr>
16. <tr><td><input name="" type="submit" value="Submit file"></td></tr>
17. </table>
18. </form>
19. You can either use any <b>.htm</b> or <b>.doc</b> file you have on your client.
20. </center>
21. </body>
22. </html>
```

The form tag appears on Line 10. In this form, 2 attributes are used: action, which specifies the PHP script for processing the submitted information, and method, which determines the way the information should be transferred. Note that we must use the post method, which will be explain
in a later session. We also postpone the discussion of the PHP script, acknowledge.php, to the next session.

The form type of content is determined by the <input> tags in Lines 12 -16. All input tags have 2 attributes in common, the name and the type of input. As long as the name is not yet used, it can be chosen quite freely (avoid special characters and blanks). Available values of the type are text, password, radio, checkbox, file, image, and submit. For type="file" there is also a third attribute, enctype. For all types that are optional there are attributes that can determine the size of the fields for giving answers.

Input tags of type="submit" are special. They do not require any name specified, but you can put text on the submit button by means of the value attribute.

The form script can contain other tags than <input>, such as the <select> tag to create menus, <textarea> for creating an area into which the user can provide a longer text, and others.

Frames

The last feature of HTML we want to cover in this introduction is frames. In the examples above, we have developed a menu page from which we can select the special feature we want to be demonstrated. However, after the first demonstration, we have to use the Back button to find the menu again. We therefore need a way to divide the screen into 2 windows, one showing the menu permanently and the other displaying the topic selected for demonstration.

The frame feature of HTML permits us to divide the screen into 2 or more windows, all visible and active at the same time. This feature uses 2 tags, <frameset> and <frame>. The page below generates the effect we want.

1. <DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
2. <html>
3. <head>
4. <title>frame.htm</title>
5. </head>
6. <frameset cols="20%,*" frameborder="yes" border="1" framespacing="0">
7. <frame src="table2.htm" name="leftFrame">
8. <frame src="blank.htm" name="mainFrame">
9. </frameset>
10. </html>

Note the difference in the <!doctype> tag from those used in previous HTML pages and that the <body> tag is not included.

Lines 6 - 9 specify a frame set. The <frameset> tag has 4 attributes:, cols, with the value "20%,*" divides the width of the client's screen into 2 windows by a vertical border, assigning
20% of the screen to the left window and the rest to the right window; **frameborder** and **border**, specifying a visible border of size 1, and, finally, **framespacing**, which is set to 0.

Inside the frameset block there are 2 `<frame>` tags, one for each window. They both have 3 attributes, which specify the **src**, i.e. the file to provide content to, and the **name** of the respective window. This page generates the 2 windows and their initial content (the right window is empty because it is generated by **blank.htm**). To understand how the further content of the windows is created, we need to look at a modified version of **table.htm** called **table2.htm** (only the part within the `<body>` block is reproduced):

1. `<center>`
2. `<h2><font color="#0000FF"><b>Menu for the example options</b></font></h2>`
3. `<p>This example illustrate the basic features of HTML which are:</p>`
4. `<table>`
5. `<tr><td><a href="blank.htm" target="mainFrame">1. Blank page</a></td></tr>`
6. `<tr><td><a href="text.htm" target="mainFrame">2. Text page</a></td></tr>`
7. `<tr><td><a href="link.htm" target="mainFrame">3. Page with link</a></td></tr>`
8. `<tr><td><a href="image.htm" target="mainFrame">4. Page with picture</a></td></tr>`
9. `<tr><td><a href="table.htm" target="mainFrame">5. Page with table</a></td></tr>`
10. `<tr><td><a href="form.htm" target="mainFrame">6. Form page</a></td></tr>`
11. `</table>`
12. `</center>`

The only **difference** from the original table.htm is the inclusion of the argument **target** with value "**mainFrame**" in the `<a>` tags of Lines 5 -10. The target directs the browser to display the link in the window named **mainFrame**, i.e., the right hand window.

**CSS, JavaScript, and XML**

The tool case for preparing web documents contains a number of useful objects. Close to **HTML** are **Cascading Style Sheets (CSS)**, **JavaScript**, and **eXtensible Markup Language (XML)**.

**CSS** was developed for use with **HTML** and introduced in 1996, and is implemented in most browsers.
Session 2: Dynamic applications in PHP

Dynamic web sites

The static model of the web interaction is based on a set of pre-developed static web pages stored on a host server, and in 3 basic steps:

1. A client sends a request for a web page to the host.
2. The host sends a copy of the requested page to the client.
3. If desired, points 2 and 3 are repeated for new pages.

A node in the web which manages the host tasks is called a web server. In the static model,

![Figure 2.1: The web user-server communication](image)

**Figure 2.1: The web user-server communication**

In Figure 2.1, the host has no ability to analyze the request and adjust the response accordingly. The response is a requested pre-designed web page. The request-response exchange is therefore called static. However, the exchange protocol used, HTTP, provides possibilities for some additional items of information to be sent to the host with the request without any instructions from the client. In the same way, the responding host can include additional information with the response, usually hidden for the receiver. The host has also capabilities to forward messages to other programs beyond the web server for additional processing. These possibilities for information processing behind the scene make it possible to create the additional functionality.

We shall use the term dynamic web site to emphasize that we are not concerned with a simple set of web pages with HTML tags, but with applications in which the pages returned to the client can be dynamically adjusted to fit the individual requests of the client. This course can serve as a first illustration of a dynamic web site. You have already experienced that when you submitted your personal access code entering this course, the system became accessible to you. If you had submitted an invalid access code, however, the host would have sent you a message adjusted to an unacceptable access code it received from you. The system must be able to compare your identity with a pre-loaded list of authorized identities. You will soon also see that if you try to go on to the next session before the time it is officially opened, you will receive a message.
regretting that the session is not yet open. When opening the date is passed, and you have passed the test at the end of the session, the system will respond by giving you access to the session. The system must be able to compare your request with its clock time and with your recorded test performance. If a student has not yet completed the required test, the host will return a message that the test must be done before the student can proceed. This means that the system must be able to keep track of your previous interactions.

Important characteristics of a dynamic web site are the ability to authenticate you, i.e. to verify your identity, to record your performance history, to react on the time for the request, to keep track of your interactions from you start a session and to its end, and sometimes even from session to session. The dynamic web site can be summarized by Figure 2.2.

![Figure 2.2: Diagram for a dynamic application](image)

**CGI and PERL**

The first step toward dynamic web pages is the possibility for a remote client to request the execution of a process at the host. Use of the FORM tags of HTML requires, for example, that the server can perform a processing of the data submitted on the form. A program must exist for
this purpose at the host site, and the web server must be able to communicate with this. We shall refer to such a program which supplements the HTML pages as a script.

The Common Gateway Interface, CGI, is a protocol specifying how certain scripts can communicate with web servers. One of the most frequently used tools for creating such scripts is the script language PERL. A PERL script stored in the host computer can be supplied with data from a request, for example sent by a HTML FORM page. The script can be designed to perform a variety of tasks such as save and retrieve data from a database, update a log, keep track of visitors, run a course, etc. It can also be designed to perform its task and then leave the result to the web server, which returns a web page generated by means of the script to the requesting client. Programming languages as C, C++, C# and JAVA can also be used for creating scripts. One reason for the popularity of PERL is that scripts programmed in PERL can be ported from one operating system to another with no or little modification.

Applications Program Interfaces

A PERL-CGI application is time-consuming because PERL scripts must be loaded, executed and unloaded each time they are used like interpretive programs, and do not offer the flexibility which may be required.

To improve this situation, Application Servers were developed. An application server is a service operating behind the web server. It processes script code, which the web server does not understand, and returns the results to the web server for returning to the requesting client. The applications server is a resource of permanently loaded executable programs, and is referred to as an Applications Program Interface, API. The advantages of using an API compared with the earlier interpretative programs are increased speed and flexibility because no loading and interpretation is needed. The disadvantage is that the API programs must be implemented and compiled separately for each type of operating system, and requires more memory space.

PHP Language

The well-known API tools include the ASP and ASP.NET from Microsoft, the open source system PHP, iHTML from Inline Internet Systems, and ColdFusion MX from Adobe. In this course, we are leaving the comparisons between the tools to evaluators and sales people, and concentrate on PHP because it is an open source tool, easily available and supported by a large community of users. PHP was introduced in 1995 as Personal Home Pages. Since then, PHP has been developed to a very powerful tool for creating dynamic web sites.

The language, by means of which we design our scripts, is the PHP Language. Files, in which these scripts are saved, are recognized by their extensions, .php. You are referred to the section Software to get detailed instructions for installing necessary software on your own PC to be able to develop and test your dynamic sites.

In the previous paragraph, the advantage of using a web API instead of an interpretive approach was emphasized. PHP was introduced on the market in 1995. It started out as a scripting language based on CGI. Later, the API was developed. The current version is PHP 5 which is a
powerful system with an embedded database system, SQLite. Be certain that you have the PHP 5 version installed.

**PHP** is widely used by individuals and enterprises among which there exist an active interchange of software and experience.

**Approach in this course**

Most courses and textbooks on programming and scripting languages, start with the introduction of the language syntax. We shall take another approach, *learning by examples*, i.e. in each session we shall introduce a set of problems with their live solutions, and explain the syntax required by the examples. In parallel with studying the examples and the text, the student should read the relevant parts of the course textbook to make certain that (s)he will acquire the precise details of the language syntax.

**Simple example**

Imagine an application requiring registration of some personal data from visitors and which should be returned as confirmation of accepted data. This simple task cannot be done by use of HTML alone because the response must be adjusted to the submitted data. Figure 2.3 outlines

![Diagram for the registration](image-url)
the application in a diagram. The diagram indicates how the communications between the user and the host pass through the web server to the PHP scripts because the server cannot process the indata but is needed to return the web pages to the user for display. To summarize the task:

1. Design a HTML form for acquiring the required data
2. Develop a PHP script for returning a confirmation of received data

The development of a HTML form may result in a typical file as:

```html
<html>
<head>
<title>Registration</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<center>
<h2><font color="blue">Registration form</font></h2>
<form action="confirm.php" method="post">
<table>
<tr><td>First name:</td><td><input name="FirstName" type="text" size="30"></td></tr>
<tr><td>Last name:</td><td><input name="LastName" type="text" size="30"></td></tr>
<tr><td>Email address:</td><td><input name="Email" type="text" size="28"></td></tr>
<tr><td></td><td><input name="" type="submit" value="Submit"></td></tr>
</table>
</form>
</center>
</body>
</html>
```

The index.htm page code specifies 3 text input fields and a submit button. There is nothing special with this code. Note that the statement in Line 10 has an ACTION attribute with value confirm.php implying that the control is transferred to a PHP file. This tells us that a dynamic application consists of a mixture of .htm and .php files. By convention in this course, the first file of any example is either named index.htm or index.php.

The purpose of the next file, confirm.php, is to instruct the server to return a confirmation for the received data. Except for a comment line including the name of the file, it contains a short PHP script. A PHP script is recognized by the start tag <!-- and the end tag -->. The script is short, but introduces several basic PHP Language characteristics.

```php
<?php
print("<center>\n<h3><font color=blue>The following data have been received:</font></h3>\n" . htmlspecialchars($_POST['FirstName']) . "\n<br>
" . htmlspecialchars($_POST['LastName']) . "\n<br>
" . htmlspecialchars($_POST['Email']) . "\n</center>\n";
```

1. Design a HTML form for acquiring the required data
2. Develop a PHP script for returning a confirmation of received data

The development of a HTML form may result in a typical file as::

```html
<html>
<head>
<title>Registration</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<center>
<h2><font color="blue">Registration form</font></h2>
<form action="confirm.php" method="post">
<table>
<tr><td>First name:</td><td><input name="FirstName" type="text" size="30"></td></tr>
<tr><td>Last name:</td><td><input name="LastName" type="text" size="30"></td></tr>
<tr><td>Email address:</td><td><input name="Email" type="text" size="28"></td></tr>
<tr><td></td><td><input name="" type="submit" value="Submit"></td></tr>
</table>
</form>
</center>
</body>
</html>
```

The index.htm page code specifies 3 text input fields and a submit button. There is nothing special with this code. Note that the statement in Line 10 has an ACTION attribute with value confirm.php implying that the control is transferred to a PHP file. This tells us that a dynamic application consists of a mixture of .htm and .php files. By convention in this course, the first file of any example is either named index.htm or index.php.

The purpose of the next file, confirm.php, is to instruct the server to return a confirmation for the received data. Except for a comment line including the name of the file, it contains a short PHP script. A PHP script is recognized by the start tag <?php and the end tag ?>. The script is short, but introduces several basic PHP Language characteristics.

```php
<?php
print("<center>\n<h3><font color=blue>The following data have been received:</font></h3>\n" . htmlspecialchars($_POST['FirstName']) . "\n<br>
" . htmlspecialchars($_POST['LastName']) . "\n<br>
" . htmlspecialchars($_POST['Email']) . "\n</center>\n";
```
These are:

- Each PHP statement line ends with **semicolon**.
- **print()** functions are used to return a message to the client.
- **$_POST[]** elements are used to refer to values submitted in a form.

Let us first explain the **$_POST[]**. All variables in PHP are recognized by $ as their first character of their name. A name followed by [...] indicates an array, and the content of the square parentheses refers to the key for the element the value of which is contained in the expression. The array **$_POST[]** is an auto-global array in which all variable values submitted to the server in a HTML FORM tag with METHOD="post" are stored during the session. PHP automatically stores these variables in the array and they are available to all parts of the application for the remaining time of the session.

Note that in PHP, the elements of the returned HTML page must be enclosed as arguments surrounded by double quotes in the print functions. The set of **print()** functions in the script, is PHP's way to send the return data to the web server, which converts these to an ordinary HTML page. Look up View ->Source in your browser when running the example to see how the return message is returned as an HTML page.

Note that this simple example is impossible to execute without a system behind the web server.

**Guessing example**

We shall advance our PHP demonstration by selecting a well known type of arithmetic guessing game which you may already have met in other contexts. The task of the game is to guess the sum of all integers up to a random number. The server will, on your request, first return a page containing an upper limit number, one box to fill with your name, another to fill with the sum you guess, and a button for submitting the guess to the server. When received, the server will compare your guess with the computed sum, and return a personalized result page to you. Figure 2.4 display the interaction between the server and the client. Even though this example is by topic quite different from the previous the general application structure is similar.

The dynamic features of this scenario are interesting. On request, the client is identified, and the upper limit number is computed and memorized by the server. When the response is accepted, the server recognizes the client, compares her/his answer with the stored computed sum, and reports back to the client with a named response. In other words, the server is able to combine two (multiple) requests from the same client, and adjust the response to the data received from the client.
Now, let us develop the solution to this problem. We obviously need:

1. A PHP script to generate an upper limit number and compute the corresponding sum of integers from 1 up to the limit, and to compose a HTML form to return to the client adjusted for the upper limit number selected.
2. A second PHP script to compare the content of the returned form and the previously computed sum, and compose a relevant response.

The first script is a mixture of 2 PHP blocks and a HTML page. It can look as follows:

1. <!-- index.php -->
2. <?php
3. srand();
4. $_SESSION['randval']=rand(10,100);
5. ?>
6. <center>
7. <h2><font color="Red">Guess!</font></h2>
8. <form action="response.php" method="post">
9. <table>
10. <tr><td>My name is:</td> <td><input type="text" name="name"></td></tr>
11. <tr><td>I guess the sum of all integers from <b>1</b> to <b><?php print("$_SESSION[randval] "); ?></b> is:</td> <td><input type="text" name="guess"></td></tr>
12. <tr><td></td><td><input type="submit" value="Submit"></td></tr>
13. </table>
14. </form>
15. </center>

Since this file contains PHP code, it must be named index.php. The first PHP script block is on Lines 2-5. Line 3 is a PHP function used to get a random seed for the function on the next line. The consequence of this is that if you repeat the application it is most likely you will get a new random upper limit numbers.

Functions in PHP always consist of a name followed by a pair of parentheses. The parentheses may be empty, as in this case, or contain one or more arguments. The srand() function requires no arguments. It is not necessary to assign the results of this function to any variable because we shall not need to refer explicitly to the seed.

Line 4 assigns the outcome of a second built in function, rand(10,100). This function, which requires 2 arguments, generates a random integer between 10 and 100, and assigns the result to the variable, $_SESSION['randvalue']. From the previous example we know that the $ means that it is a variable, and the [] indicates that the variable is stored in an array.

The $_SESSION[] array is used to store all variables which we want to access at different occasions during a session. Like the the $_POST[] array, the $_SESSION[] array variables are superglobal variables (It is assumed that you during configuration set the session_auto_start to ON in the php.ini after installation). The session variables are kept for a default period up to 1440 second (24 minutes) after which the session cookie expires. The server's recognition of a client is obtained by means of cookies, which is returned with the server's response to the first
request from the client, and then connected to all requests from the client to the server within the session.

Following the first PHP block, is a HTML form with an embedded PHP block, `<?php print("<b>$_SESSION[randval] </b>") ?>` embedded in Line 11. The reason for including this script line in the middle of an HTML expression is that we want to include the PHP variable `$_SESSION[randvalue]` to be displayed for the client. Note that within double quotes, as in the print argument, single quotes are not used around the array keys, e.g. in `$_SESSION[randvalue].`

The form calls upon the second script, `response.php`:

1. <!-- response.php -->
2. `<?php
3.   $sum="0";
4.   for ($count=1;$count<=$_SESSION['randval'];$count++) {
5.     $sum=$sum + $count;
6.   }
7.   if ($sum == $_POST['guess']) {
8.     echo "$_POST[name], your guess was correct!";
9.   }
10.  elseif ($sum > $_POST['guess']) {
11.    echo "Sorry, $_POST[name], your guess <b>$_POST[guess]</b> is too low, the correct sum is <b>$sum</b>";
12.  }
13.  else {
14.    echo "Sorry, $_POST[name], your guess <b>$_POST[guess]</b> is too high, the correct sum is <b>$sum</b>";
15.  }
16. `>

The Lines 3 - 5 computes the correct sum associated with the generated upper limit integer, `$_SESSION[randval]` by looping through a for loop with an index variable named `$count` which is increased by 1 using the incremental operator `++`, and for each loop the `$sum` is increased by the current index number.

Lines 7 - 16 contain a test of the guess submitted (NB. Note that the comparison operator `===` is used in the test condition), and return an answer to the client. Three alternatives are possible: Line 8. will be sent as an HTML page to the client if the sum guessed is correct, elseif the guess is less than the correct sum, Line 11. will be executed, and, finally, if the guess is too high, Line 14. is used for response to the client.

The last script illustrates how PHP can solve dynamic tasks by using$_POST[] and $_SESSION[] variables. Both these arrays contain global variables, i.e. variables which are persistent during the client's session, an important requirement for dynamic application development.
Session 3: Dynamic application without database

Market research

In this session, we shall study how a rather dynamic application can be designed without any database support. The scenario we shall use is online collection of data for market research. The marketing problem concerns 2 products, A and B. We are interested in measuring consumers' relative preferences for the two competing products. However, we have a suspicion that the respondents may have a tendency to vote for the product listed first. To eliminate this effect, we want to randomize the sequence, i.e., AB and BA. The persistence of the preference is another question we want to study. For this reason, we want the respondents to vote for their preference a second time, e.g. a week, after the first vote. To attract consumers to vote, those who complete the 2 votes are eligible for participation in a lottery.

A file of responses must be built in which the 2 votes of the individual consumers can be connected by mean of a unique identifier for comparing responses as well as a file with name and addresses for those who are eligible for lottery participation.

Since this is a course focusing on design and development of dynamic web sites, the important questions about how to obtain representative participants and how many, are left for the statisticians. Also the questions about the evaluation of the reliability of the results are considered outside our scope in this course.

System design

Figure 3.1 gives an outline for how we want to solve the task stated above. There are 2 application parts which are both connected to the same 2 data files. The implementation will consist of .php, .htm and .txt files demonstrating how it is possible to mix different types of files in the same application.
We start by listing the files we shall need for the application:

**Market_research:**
- index.php
- prepare.php
- save.php
- form3.htm
- save3.php

**Market_analysis:**
- index.htm
- report.php
- report2.htm

**Common text files:**
- responses.txt
- addresses.txt
We use the convention introduced in the first session, and name the first file of each application part `index.php`. This convention gives us the advantage that we can open the application by calling the folder in which all the files reside. The file list reflects 3 sets of files, the user module, `market_research`, the administration module, `market_analysis`, and the data files as outlined in Figure 3.1. In addition, some global arrays of variables (i.e. `$_POST[]`, `$_SESSION[]` and `$_COOCKIE[]`) exist for creating persistency during the application sessions.

The `.txt` files do not exist initially, but is generated when the first data are collected. They are used for permanent saving of data representing the persistency from one session to another.

**Market research**

Figure 3.2 gives a simplified picture of the Market Research part of the application. The index page is used to send the users instructions about what to do. Since there are 3 questionnaires to be completed by the participants in the research survey, we define in Line 4 a variable, `$_SESSION['marker']`, to keep track of which questionnaire is the current. This variable is an element of a global array used for making variable values persistent for access in several scripts during a session. Line 3 test by means of a function `isset()` if `$_SESSION['marker']` has been defined, and if not define and set the variable to 1.
Next, observe that except for the 3 if statements, the remaining of this PHP script prepares 3 alternative displays using the print() function. Line 6 prepares the first common part of the HTML page to be returned to the client, while the Lines, 7, 10 and 13 test which questionnaire should be offered the user. Depending on the value of the marker variable, Line 8, 11 or 14 is sent with an a tag linking to the appropriate questionnaire. If you study these print statements carefully, you may be surprised by noticing that there are no double quotes around the files to which the A tags refer. Expressions already enclosed in the double quotes of a PHP statements, should not contain any double quotes.

```php
1. <!-- index.php -->
2. <?php
3. if(!isset($_SESSION['marker'])) {
4.    $_SESSION['marker']=1;
5. }
6.  print("<center><h2><font color=Blue>Market research</font></h2></center>p>This is a market research to investigate the public's preferences for Product A and Product B. If you respond and complete two questionnaires, you will be eligible to participate in a lottery. The requirements are:<p/>
7. if($_SESSION['marker']==1) {
8.    print("<b>Request, complete and submit <a href=prepare.php>questionnaire 1</a></b>";?></p>
9. }
10. if($_SESSION['marker']==2) {
11.    print("<b>Request, complete and submit <a href=prepare.php>questionnaire 2</a></b>";?></p>
12. }
13. if($_SESSION['marker']==3) {
14.    print("<b>Request, complete and submit <a href=form3.htm>questionnaire 3</a></b>");</p>
15. }
16. print(" <p>The 2 first questionnaires require you make a single click only before you submit your response. The third questionnaire asks for you e-mail address for notification in case you become a winner in the lottery.</p><p><i>The market research sets a time-limited cookie in your browser.</i></p>
17. ?>
```

index.php does not contain any new PHP features, and we can proceed to the next script, which is the prepare.php. The purpose of this script is to prepare the 3 different questionnaires and keep track of which should be served.

Already the first lines introduces an important new feature, i.e. the use of cookies. A cookie is a small message sent from the server to the client browser attached to a reply to a request. The cookie is providing the receiving client with a unique identifier, a time-out specification and an identification of the server which has issued the cookie. Cookies are kept in a special list in the computer, and deleted when timed out.

Before a browser sends a request to a server, its list of cookies is checked for any cookies from the server approached. If a relevant cookie is found, it is copied and attached to the request. The server receiving a request, scans the request for cookies. When a cookie is detected, the server has received a user identification and can take advantage of any data stored about the client.
In this way, it is possible to link items in a chain of interactions between the server and an individual client. Since the server is issuing and distributing cookies, the anonymity of the client can be maintained. In our application, we ask the client to answer our questions at 2 different occasions. We can link the answers by means of a cookie without inquiring about the name or other identification from the client. Note that this assumes that the client is using the same computer and is the only user of the computer.

In Line 3 - 6, we ask if the requesting client has a cookie called user_id, and, if not, prepare in Line 4 a cookie to be returned to the client with the response to its request. The name of the cookie to be sent is user_id and we use as its unique value is the exact time obtained by the PHP function time() in Line 4, at the moment the cookie is set. We use a built-in function, setcookie(), for setting cookies. This function requires to be set as early as possible in the script, and always before any other content is sent to the client. In the function setcookie(), we specify the name of this cookie, in our application user_id, the value of the cookie, and the cookie timeout of the cookie. The cookie timeout is determined as the time when its validity ends. We use again the return value of the function time() and add the lifetime measured in seconds. In our particular application, the second preference form should be answered one week after the first at which the cookie is set and the timeout point should be more than 8 days (>60x60x24x8) later. For the purpose of illustration only, this session's example sets the lifetime of the cookie to only 60 seconds.

It is important that only one cookie is set for each visitor, and for that reason a test is made in Line 3 for the existence of the particular application cookie, $_COOKIE['user_id']. If it already is set, Line 4 and 5 are not executed.

1. <!-- prepare.php -->
2. <?php
3. if(!isset($_COOKIE['user_id'])) {
4.     $time=time();
5.     setcookie('user_id','$time', "$time" + 60);
6. }
7. rand();
8. $randval=rand(1,2);
9. $_SESSION['marker']++;
10. print("<center><h2><font color=Blue>Preference for products</font></h2></center><p>Thank you for visiting this page and expressing your opinion. Complete and submit this form. The second questionnaire should be completed one week after the first.</p> <FORM ACTION=save.php method=post>
11. <p>Please mark your preference by clicking a button. Comparing the 2 products A and B, I prefer: </p>"");
12. if($randval == "1") {
13.     print("<p><INPUT TYPE=Radio NAME=preference VALUE=A> Product A</p>
14.     <p><INPUT TYPE=Radio NAME=preference VALUE=B> Product B</p><INPUT TYPE=hidden NAME=form_type VALUE=1>"");
15. }
16. else{
17.     print("<p><INPUT TYPE=Radio NAME=preference VALUE=B> Product B</p>
18.     <p><INPUT TYPE=Radio NAME=preference VALUE=A> Product A</p><INPUT TYPE=hidden NAME=form_type VALUE=2>"");
19. }
20. print("<p><INPUT TYPE=submit NAME=SUBMIT VALUE=Submit></p>"
The last point to be mentioned is the incremental operator ++ used in Line 9, well known from other languages. This line is equivalent to the longer statement

```php
$_SESSION['marker']=$_SESSION['marker'] + 1;
```

The questionnaires expressed by `print()` functions in Line 10, 13 and 17 are served to the clients depending on the value of `$_SESSION['marker']` incremented in this way.

The returned responses from the clients are taken care of by the script `save.php`. The answers to the questionnaires 1 and 2 are saved in `response.txt`. If the file does not exist, it is established by the PHP function `touch()`. Before any file can be operated on, it must be opened by means of the `fopen()` function which requires 2 arguments, the file name and the action. There are 2 write action available, write from the beginning and append to the end of the file indicated by "w" or "a", respectively. In Line 6 the `response.txt` is opened for appending data. The `fopen()` returns a handle or reference, in this script called `$f`, and which is used in the file action function `fwrite()` in Line 7 to indicate which opened file should be acted on.

```php
if(! file_exists("response.txt")) {
    touch("response.txt");
}
$f=fopen("response.txt","a");
fwrite($f,"User id: $_COOKIE[user_id], Form type: $_POST[form_type], Preference: $_POST[preference]\n");
```

In our live example, you can return to complete questionnaire 2 immediately. In a real application, we would like to observe the preference change during a whole week, which could require some kind of script for reminding the client about the second questionnaire in a week.

The registration of the participants for the lottery, implemented in `form3.htm`, is sent to the clients when both data collection forms have been returned. It is a simple form calling `save2.php`. It returns the submitted name and email address by `METHOD="post"` in order to be easily available by `save2.php`. 

fwrite() requires 2 arguments, the file handle and a string of the items to be written to the file. In Line 7, the first argument of fwrite() is the file handler `$f` just established in the previous line, and a string of 3 name/value pairs for `User_id`, `Form type` and `Preference`, all variables from global arrays. These variable are delimited by commas. These are the 3 items in which we are interested in. Note the end of line symbol, `\n`, at the end of the string to get a line shift after each record.

In our live example, you can return to complete questionnaire 2 immediately. In a real application, we would like to observe the preference change during a whole week, which could require some kind of script for reminding the client about the second questionnaire in a week.
1. <!-- form3.htm -->
2. <center><h2><font color="Blue">Your e-mail address</font></h2></center>
3. <p>If you are eligible for participating in the lottery, i.e. you have requested, completed and submitted the two questionnaires, we need your e-mail address to notify you in case you become a winner in the lottery.</p>
4. <form action="save2.php" method="post">
5. <table>
6. <tr><td>Your name:</td><td><input type="Text" name="myname"></td></tr>
7. <tr><td>e-mail address:</td><td><input type="Text" name="myemail"></td></tr>
8. <tr><td></td><td><input type="Submit" value="Submit"></td></tr>
9. </table>
10. </form>

The save2.php script is very similar to the already discussed save.php. It resets the marker to the initial value 1 and thanks the client for his/her participation.

1. <!-- save2.php -->
2. <?php
3. if(! file_exists("address.txt")) {
4. touch("address.txt");
5. }
6. $a=fopen("address.txt","a");
7. fwrite($a,'user id: $_COOKIE[user_id], Name: $_POST[myname], Email address: $_POST[myemail]\n ');
8. $_SESSION['marker']=1;
9. ?>
10. <center>
11. <p><b><font color="blue"> Thank you for your participation. You will be included in the lottery.</font></b></p>
12. </center>

Market analysis

The scripts so far have been aimed at collecting the preferences of the participants of the market research survey. However, we need also to have tools for retrieving the collected data for analysis. The second part of the application in this session is named Market_analysis (In fact, it is not an analysis, but a data retrieval): Figure 3.3 gives an overview of this part of the application. It starts with a simple HTML page, index.htm, which offers 2 options. The first activates the script report.php which prints the text file response.txt. the second starts report2.php which prints the text file address.txt.
The index.htm is an ordinary HTML page which links the 2 alternative scripts, report.php and report2.php.

1. <!-- index.htm -->
2. <html>
3. <head>
4. <title>index.htm</title>
5. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6. </head>
7. <body>
8. <center>
9. <h3><font color="#0000FF">Data results</font></h3>
10. <p>There are 2 results available from market research online data collection:</p>
11. <table>
12. <tr><td>1. <a href="report.php">Results</a> from the research</td></tr>
13. <tr><td>2. <a href="report2.php">List</a> of qualified contestants</td></tr>
14. </table>
15. </center>
16. </body>
17. </html>

The report.php script lists the data as recorded by save php. This time, we open the file for reading by the parameter "r". By means of a while block, Line 4-7, using the function feof($r)
as the core of the condition, the file is read line by line until the end of file is appearing. Each line is retrieved by the function fgets(), and sent for display.

1. <!-- report.php -->
2. <?php
3. $r=fopen( '../market_research/response.txt', 'r');
4. while(!feof($r)) {
5. $line=fgets($r);
6. print("$line<br>");
7. }
8. print("<center>
9. <p> <a href=admin.htm>Return</a> to menu.</p>
10. </center>");
11. ?>

The second script, report2.php, differs only in the specification of the file, address.txt, to be read. The lines read by this script have the content as written by save2.php.

1. <!-- report2.php -->
2. <?php
3. $r=fopen( '../market_research/address.txt', 'r');
4. while(!feof($r)) {
5. $line=fgets($r);
6. print("$line<br>");
7. }
8. print("<center>
9. <p> <a href=admin.htm>Return</a> to menu.</p>
10. </center>");
11. ?>

We can finalize the example by a simple menu script in HTML to be able to select between the execution of the research and displaying the results.

This application is characterized by reading the records serially to the server during the collection of data, and retrieving the results in the same order from the server after the collection has been completed. In many applications data already saved are updated in a random order as well as requested in a non-serial order. For such application, use of a database will usually be a better solution and will be the topic of the next sessions.
Session 4: Introducing the SQLite database

Dynamic applications and databases

In previous sessions, we have studied examples of dynamic applications in which have made use of session variables to adjust dynamically to data provided by the client previously, and use of files to store data permanently. The use of files can have serious drawbacks since reading or updating a record may require that the whole file must be searched.

Using a database instead of a file or a set of files makes it possible to retrieve or update a single record. A database has usually its own software, the Data Base Management System, which operates on the data. To connect PHP to the database, software called ODBC. is usually needed. The most popular database used in connection with PHP is MySQL, which is another open source and free software. Commercial database software frequently used with PHP are POSTGRESS, ORACLE and SyBASE. All these database systems require separate installation.

SQLite is a new system based on a library of C programs and implements a database system without any configuration requirements. The database system can also be used as a self-contained system. SQLite is reported in general to be faster than MySQL and PostgresSQL for smaller and medium size PHP application and excellent for learning the fundamentals of web applications with supported by databases.

SQLite comes embedded in PHP 5. No additional installation is required, and no special drivers needed for connecting to the database. In this course, we shall take advantage of these facts and use SQLite in our further work.

SQLite

The SQLite database system responds to SQL92 specifications (a few features are not implemented) and is said to reside between MySQL and PostgresSQL in functionality. The database is kept in a single disk file, and can be shared by several servers. The size of the database can be considered limited only by disk capacity of the host server. In contrast to most other database systems, SQLite is type free, i.e. it associates data type with the data itself instead of the declared column data type. A column declared for TYPE text, can for example accept a BLOB (Binary Large Object). Most SQLite operations are faster than corresponding operations on other databases.

Creation of a reference database to you personal library.

Most people buy and collect books. The collection can contain books belonging to different categories such as poetry, prose, fiction, science fiction, historic and contemporary documentaries, information systems, web applications, databases, theoretical and applied
research, and probably many other genres. From time to time, we want to return to our books to search for a particular book or set of books. Sometimes, when we recall the author's name or the title of the book the search may be easy, in other situations we may only recall certain aspects discussed which make the search more difficult.

As an introductory example to databases, we shall create a database system which we can populate with the necessary data about each book in our personal library. Figure 4.1 gives a graphic model for the application.

Figure 4.1: Components of a personal library database

We shall need the following files:

- HTM page with example menu
- HTM page recording data about a book
- PHP script for creating the database, and adding data recorded for a book
- HTML page for requesting a list of rows from the database
- PHP script for retrieving the rows and responding to the request
- HTML page for requesting data about a book(s)
- PHP script for retrieving the requested data from the database, and sending it to for display to the client
- HTML page to modify or delete a record(s) in the database
- PHP script for executing the modification/deletion

This system is fully functional, but you should consider to make your personal modifications to the design before you start recording your personal library.

**Menu page**

The *index.htm* is a very simple menu system with centered display of links to the different parts of the reference system:

1. <!-- index.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3. "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
7. <title>Untitled Document</title></head>
8. <body>
9. <center>
10. <h2><font color="blue">Menu for the book database</font></h2>
11. <p>The following menu lists the alternatives available for the database <b>books</b>.
12. </p>
13. <table>
14. <tr><td><a href="add.htm">Insert</a> book data to a database row.</td></tr>
15. <tr><td><a href="list.php">List</a> all rows in the book database.</td></tr>
16. <tr><td><a href="search.htm">Search</a> for a row of data in the database.</td></tr>
17. <tr><td><a href="update1.htm">Update</a> a row in the database.</td></tr>
18. <tr><td><a href="delete.htm">Delete</a> a row from the database.</td></tr>
19. <tr><td><a href="remove.htm">Remove</a> database content.</td></tr>
20. </table>
21. </center>
22. </body>
23. </html>

To obtain a nicer visual impression, the links are embedded in a table. The result is shown in *Figure 4.2*
Creating and populating a database

Let us start studying the creation of a bibliographic database. There are international standards and protocols for the content required for a professional bibliographic database. In this example, we shall only extract the elements needed for a database for private use. It must obviously contain such data as the name of the author(s), the book title, the publisher's name, when printed, and the number of pages.

We may also want to be able to make a rough distinction about the categories of books. You are free to establish your own lists of category, evaluation and location codes.

1. <!-- add.htm -->
2. <center><h3><font color="#0000FF">Adding records to database</font></h3></center>
3. <p>This form is used to add a reference to a new book in the database. The database will be automatically established the first time this system is used.</p>
4. <p>You are free to develop your own categories, evaluation and location codes. The location code can be a combination of text and a number, for example Bookshelf_a_33.</p>
5. <form action="add.php" method="post">
6. <table>
7. <tr><td>Name(s) of author(s) : </td><td><input name="author" type="text"></td></tr>
8. <tr><td>Title of book : </td><td><input name="title" type="text"></td></tr>
9. <tr><td>Publisher: </td><td><input name="publisher" type="text"></td></tr>
10. <tr><td>Year of publication : </td><td><input name="year" type="text"></td></tr>
11. <tr><td>Number of pages: </td><td><input name="pages" type="text"></td></tr>
12. <tr><td>Category: </td><td><input name="category" type="text"></td></tr>
13. <tr><td>Date read : </td><td><input name="dateread" type="text"></td></tr>
14. <tr><td>Evaluation: </td><td><input name="evaluation" type="text"></td></tr>
15. <tr><td>Book location : </td><td><input name="location" type="text"></td></tr>
The 9 named values in the form block Line 5- 16 are submitted to the server for processing by means of the add.php script. The form appears as in Figure 4.3.

The file add.php is our first step into the world of SQLite. Let us first select a name, books, for our database, and booktable as the name of the single table in this simple database.

The first sqlite statement in Line 3 is very powerful in the sense that it opens the database books for access if it exists, or creates and opens the database if no database named books exists. It outputs a reference/handle, $db, to the database for future use. The next sqlite statement is a query statement which requires a reference to the database, and an SQL statement as arguments. In Line 4, the second argument is a SELECT statement which retrieves from a table called sqlite_master maintained by the sqlite system and keeping track of all tables created within the referenced, $db database. The function produces an array, $a, with one row with information per existing table. From Line 5 we obtain the number of rows, $b, i.e. number of tables that exist in books. Per design, we know that the only possibilities are 1 if the table booktable exists, or 0 if that table does not exist.

If 0 tables exist, the booktable is created by the sqlite-query function with the SQL statement CREATE TABLE in Line 7. Following the specified table name in parenthesis are the table's columns specified with their name and type. In our application we have 9 columns, all of TYPE TEXT. They are preceded by a column with a unique id of TYPE INTEGER PRIMARY KEY which automatically assigns a unique id to each row added to the table.
In the fourth \texttt{sqlite\_query} function in \textbf{Line 9}, the 9 variable values received from the client are \texttt{added} to the \texttt{booktable} by means of an \texttt{INSERT} statement. Note that it is very important that the syntax is correct, if not the statement will not work. In particular, remember to enclose all values of type \texttt{TEXT} in \texttt{single quotes}, and do \texttt{NOT} use quotes within the \$_\texttt{POST}[] because the \texttt{INSERT} statement itself is enclosed by double quotes. Be also certain that the elements in \texttt{booktable(..)} matches the values in \texttt{VALUES(..)}.

\textbf{Listing the content of the database}

The request for a list of the rows in the database does not require any additional data, and it is therefore possible to call directly from the menu, index.htm, to the php script generating and returning the list to the client.

\begin{verbatim}
1.  <!-- list.php -->
2.  <?php
3.  print("<center>");
4.  if(! file_exists("books"){
5.    print("<h2><font color=red>Database does not exist.</font></h2>");
6.  }
7.  else {
8.    $db = sqlite_open("books");
9.    $a = sqlite_query($db, "SELECT name FROM sqlite_master WHERE type='table' AND name='booktable'");
10.   $b=sqilte_num_rows($a);
11.   if ( $b == 0) {
12.     print("<h2><font color=red>The booktable is empty. </font></h2>");
13.   }
14.  else {
15.     print("<h2><font color=Blue>List of panel members</font></h2>
16.     <table border>
17.     <tr><b><Th>Id<Th>Author</Th><Th>Title</Th><Th>Publisher</Th><Th>Year</Th><Th>Pages</Th><Th>Category</Th><Th>Date read</Th><Th>Evaluation</Th><Th>Location</Th></b></tr>
18.     $r=sqlite_query($db, "SELECT * FROM booktable ORDER BY Id ASC");
19.     while ($row = sqlite_fetch_array($r)) {
21.       </tr>");
22.     }
23.     print("</table>");
24.   }
25. }
26. print("</p>");
27. print("<a href=index.htm>Return</a> to menu.");
28. print('<center>');
29. ?>
\end{verbatim}
In this operation, we need a message returned in case the database does not exist. This is taken care of by **Line 4-5** which instruct the server to test if the database, implemented in file **books**, exists, and if not to return a message that no database exists. On the other hand if the database exist, the next step is to test if **booktable** is empty demonstrated in **Lines 9-11**.

If the booktable is not empty, a list of the book references can be returned, in a table format as specified by **Lines 15-23**. The **sql_query** function in **Line 19** uses an **SQL SELECT** statement to retrieve all rows from booktable and order them by id value in ascending order. The output is referred to by **$r**. The **while** loop in **Lines 20-22**, fetches one row each loop as an array called **row**. The elements of this array, i.e. row[0], row[1], row[2] and so on are then printed in **Line 21**. See **Figure 4.4**.

**List of book references**

<table>
<thead>
<tr>
<th>Id</th>
<th>Author</th>
<th>Title</th>
<th>Publisher</th>
<th>Year</th>
<th>Pages</th>
<th>Category</th>
<th>Date read</th>
<th>Evaluation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trachtenberg, Adam</td>
<td>Upgrading to PHP 5</td>
<td>O'Reilly</td>
<td>2004</td>
<td>327</td>
<td>Scripting</td>
<td>December 2004</td>
<td>Good</td>
<td>Self A</td>
</tr>
<tr>
<td>2</td>
<td>Sklar, David</td>
<td>Learning PHP 5</td>
<td>O'Reilly</td>
<td>2004</td>
<td>348</td>
<td>scripting</td>
<td>July 2005</td>
<td>Does not satisfy my criteria</td>
<td>Shelf B</td>
</tr>
<tr>
<td>3</td>
<td>Sklar, D. and Trachtenberg, Adam</td>
<td>PHP Cookbook</td>
<td>O'Reilly</td>
<td>2002</td>
<td>608</td>
<td>scripting</td>
<td>September 2005</td>
<td>Incomplete</td>
<td>Shelf B</td>
</tr>
<tr>
<td>4</td>
<td>Zandstra, Matt</td>
<td>Teach Yourself PHP in 24 Hours</td>
<td>SAM2</td>
<td>2002</td>
<td>505</td>
<td>scripting</td>
<td>April 2004</td>
<td>Elementary</td>
<td>Shelf B</td>
</tr>
</tbody>
</table>

**Figure 4.4: List of book references**

**Searching the database for a book reference.**

In a large collection of books, it can be difficult to decide which book is relevant for a particular situation and perhaps also find its physical location. We need a search function. The link to searching of the menu in **index.htm** points to **search.htm**:

1. <!-- search.htm -->
2. <!--DOCTYP HTML PUBLIC "//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
3. <html>
4. <head>
5. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6. <title>Untitled Document</title>
7. </head>
8. <body>
9. <center>
10. <h2><font color="blue">Search for a reference to a book.</font></h2>
11. By specifying author, title, publisher, publishing year, category, and keywords the application can retrieve relevant book reference(s) if any.</p>
The search.htm is an ordinary form page but has the default value unspecified for each variable. The default value(s) must be changed to the value associated with the book(s) searched. If for example web applications is used as categories for books in this field, and we want to localize books published in 2005 in our library, Year of publication should be changed to 2005, and Category to web applications. See Figure 4.5.

![Search for a reference to a book.](image)

**Search for a reference to a book.**

By specifying author, title, publisher, publishing year, category, and keywords the application can retrieve relevant book reference(s) if any.

![Search form](image)

**Figure 4.5: Search for a publication**
When the request is submitted, it calls for `search.php` to process the request.

1. <!-- search.php -->
2. <?
3. php
4. print("<center>");
5. if (! file_exists("books")){
6. print("<center><h2><font color=red>Database does not exist.</font></center>");
7. exit();
8. }
9. $db = sqlite_open("books");
10. $a = sqlite_query($db, "SELECT name FROM sqlite_master WHERE type='table' AND name='booktable'");
11. $b = sqlite_num_rows($a);
12. if ($b == 0) {
13. print("<h2><font color=red>The database is empty.</font></h2>");
14. } else {
15. print("<h3><font color=blue>List of requested rows.</font></h3>");
16. <table border>
17. | Id | Author | Title | Publisher | Year | Pages | Category | Date read | Evaluation | Location |
18. $r = sqlite_query($db, "SELECT * FROM booktable WHERE (author='$_POST[author]') | (title='$_POST[title]') | (publisher='$_POST[publisher]') | (year='$_POST[year]') | (pages='$_POST[pages]') | (category='$_POST[category]') | (dateread='$_POST[dateread]') | (evaluation='$_POST[evaluation]') | (location='$_POST[location]')");
19. while ($row = sqlite_fetch_array($r)){
21. }
22. }
23. print("</table>");
24. print("</p></p>");
25. print("<a href=index.htm>Return</a> to menu.");
26. print("</center>");
27. ?>

As in the scripts above, the existence of the database and a non-empty `booktable` are tested in Lines 2-13. In an `sqlite_query()` in Line 18, retrieval of a row(s) where at least one of the variable values of the above form matches the corresponding column in the database. Since the default values were `unspecified` and the probability for a row with the string 'unspecified' as value in any columns is insignificant, this value has no impact on finding rows we request. A search result is illustrated in Figure 4.6.
### List of requested rows.

<table>
<thead>
<tr>
<th>Id</th>
<th>Author</th>
<th>Title</th>
<th>Publisher</th>
<th>Year</th>
<th>Pages</th>
<th>Category</th>
<th>Date Read</th>
<th>Evaluation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Sklar, David</td>
<td>Learning PHP 5</td>
<td>O'Reilly</td>
<td>2004</td>
<td>348</td>
<td>scripting</td>
<td>July 2005</td>
<td>Does not satisfy criteria</td>
<td>Shelf B</td>
</tr>
</tbody>
</table>

[Return to menu.]

---

**Figure 4.6: Response to a request**

---

**Updating book references in the database**

In our book reference library, there may be a need for changing or updating a row because of typos, incorrect data, revaluation of the referenced book, etc. Our solution to this task requires 1 HTM page and 2 PHP scripts.

The **update.htm** is an ordinary page which requires that you have the id of the row reference.

```html
1. <html>  
2. <head>  
3. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">  
4. <title>Untitled Document</title>  
5. </head>  
6. <center>  
7. <h2><font color="blue">Update a row in the book database</font></h2>  
8. </center>  
9. <center>  
10. <form action="update1.php" method="post">  
11. <input name="id" type="text"> <input name="" type="submit" value="Submit"> </form>  
12. </center>  
13. </center>  
14. </center>  
15. </center>  
16. </body>  
17. </body>  
18. </html>
```

There is nothing new in this page, an example of which can be seen in **Figure 4.7**.
Lines 12-13 call for processing of an row id by the script update1.php:

1. <!-- update1.php -->
2. <?php
3. print("<center>");
4. if(! file_exists("books")){
5. print("<h2><font color=red>Database does not exist.</font><form></form><h2>");
6. exit();
7. }
8. $db=sqlite_open('books');
9. $a = sqlite_query($db, "SELECT name FROM sqlite_master WHERE type='table' AND name='booktable'");
10. $b=sqlite_num_rows($a);
11. if ( $b == 0) {
12. print("<h2><font color=red>The database is empty. </font></h2>");
13. }
14. else {
15. $id=$_POST['id'];
16. $r=sqlite_query($db,"SELECT * FROM booktable WHERE id='$id'"); 
17. while($row=sqlite_fetch_array($r)) {
18. $author=$row[1];
19. $title=$row[2];
20. $publisher=$row[3];
21. $year=$row[4];
22. $pages=$row[5];
23. $category=$row[6];
24. $dateread=$row[7];
25. $evaluation=$row[8];
26. $location=$row[9];
27. }
28. print("<h2><font color=blue></font></h2>");
29. print("<h2><font color=blue><p>Correct the required attributes.</font></h2></p>");
30. print("<form action=update2.php method=post>
31. <table>
32. <tr><td>Row Id: </td><td><input name=id type=text value=$id></td></tr>
33. <tr><td>Name(s) of author(s) : </td><td><input name=author type=text value=$author></td></tr>
34. <tr><td>Title of book : </td><td><input name=title type=text value=$title></td></tr>
35. 
36. Return to menu.
The aim of this first update script is to send a form with all the data of row id to the client for inspection, correction and return to the server. Lines 16-27 retrieved the data from the database and establish local variables to be used in creating the form. Because the form must be sent in HTML format to the client, the print() is used to send the HTML tags in double quotes. PHP automatically equips the contents of these HTML tags with required quotes, and to avoid too many, we must remove all quotes in the tags. Note that one of the print() extends from Line 30 to 43.

A form as specified by the Lines 28-46 is displayed at the clients screen. (See Figure 4.8). All the
Correct the required attributes.

<table>
<thead>
<tr>
<th>Row Id:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name(s) of author(s):</td>
<td>Trachtenberg.</td>
</tr>
<tr>
<td>Title of book:</td>
<td>Upgrading</td>
</tr>
<tr>
<td>Publisher:</td>
<td>O'Reilly</td>
</tr>
<tr>
<td>Year of publication:</td>
<td>2004</td>
</tr>
<tr>
<td>Number of pages:</td>
<td>pages</td>
</tr>
<tr>
<td>Category:</td>
<td>Scripting</td>
</tr>
<tr>
<td>Date read:</td>
<td>December</td>
</tr>
<tr>
<td>Evaluation:</td>
<td>Very good</td>
</tr>
<tr>
<td>Book location:</td>
<td>Self</td>
</tr>
<tr>
<td>Submit data:</td>
<td>Submit</td>
</tr>
</tbody>
</table>

Return to menu.

Figure 4.8: Form for correcting required attributes

current variable values can be changed and the form submitted for processing by script update.php:

```
1. <!-- update2.php -->
2. <?php
3. $db=sqlite_open('books');
4. sqlite_query($db, "UPDATE booktable
5. SET
6. id='$_POST[id]',
7. author='$_POST[author]',
8. title='$_POST[title]',
9. publisher='$_POST[publisher]',
10. year='$_POST[year]',
11. pages='$_POST[pages]',
12. category='$_POST[category]',
13. dateread='$_POST[dateread]',
14. evaluation='$_POST[evaluation]',
15. location='$_POST[location]'"
16. WHERE id='$_POST[id]' ");
17. print("<center><font color=blue>");
```
This script is similar to add.php but uses the SQL UPDATE statement to change an existing row instead of the INSERT statement which adds a new row to the database. It uses the attribute SET with subsequent variable-value pairs for inserting values in the row specified by the WHERE clause. As for the INSERT statement it is very important that all values are enclosed by single quotes if they were defined as strings in the CREATE TABLE statement.

**Deleting rows in the database**

Just as we need operations for adding and updating a row, an operation for deleting a row is required. Before we can start the operation, the id of the row to be deleted must be found. It can either be obtained by the list option, or, if you know some specific characteristics, by the retrieve option. delete.htm is an HTML page for specifying the id of a row wanted to be deleted.

The page is simple, see Figure 4.9, and sends an id with a request for processing by the
Delete a row in the book database

Note the Id number of the row you want to delete and type it in the box:

![Form for deleting a row of the database](image)

Figure 4.9: Form for deleting a row of the database

delete.php script:

1. <!-- delete.php -->
2. <?php
3. $db=sqlite_open('books');
4. $r=sqlite_query($db,"DELETE FROM booktable WHERE id='$_POST[id]'");
5. print("<center><h2><font color=red>Row $_POST[id] has been removed from database books.</font></h2>");
6. print("<p></p><a href=index.htm>Return</a> to menu.</center>);
7. ?>

In an sqlite_query(), the SQL DELETE statement controls the deletion of the row specified in its WHERE clause.

Removing all content in the booktable

It is also possible to instruct the server to remove all content from the booktable. The HTML page remove.htm used is simple:

1. <!-- remove.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3. "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
7. <title>Untitled Document</title>
8. </head>
Remove all content of booktable

Note that all book references in your database will be lost by executing this operation.

Remove all content: Submit

Return to menu.

The page, the visual form is displayed in Figure 4.10, requires no explanations.

Remove database content

Note that all data in your database will be lost by executing this operation.

Remove all content: Submit

Return to menu.

Figure 4.10: Form for removal of all content

It calls upon an even shorter remove.php script.

1. <!-- remove.php -->
2. <?php
3. $db=sqlite_open('books');
4. $r=sqlite_query($db,"DROP TABLE booktable");
5. print("<center><h2><font color=red>Database content has been removed.</font></h2>");
6. print("<p><a href=index.htm>Return</a> to menu.</center>”);
7. ?>

This script introduces another new SQL statement, DROP TABLE which only requires the name of the table. Note that this operation does not remove the database books. It will still be there and will get a new table named booktable the first time add.php is processed. If you also want to get rid of the database, one possibility is to go to the folder in which it resides and delete it from the folder by means of the operating system.
Session 5: Application with SQLite database

Opinion polls

In session 2, we studied a market research example which did not require any data base backup. In this session we shall consider a similar scenario: a polling organization the aim of which is to collect the public opinion about the preferences for 5 political parties (or product brands) A, B, C, D, and E.

The organization uses a panel with a fixed number of members as basis to collect data for its services. A file with separate records for each panel members is kept in a database. Each Monday a list of all panel members with their contact addresses is retrieved and used by interviewers who, in telephone interviews, ask panel members which party the member would have voted for if there had been a public vote, or, in case of products, which product they would have purchased, that Monday. The answers are subsequently saved in the database for retrieval, computation and publication of statistics to subscribing clients each Wednesday. The panel members can be stratified by age and area in which they live.

Use of a panel gives usually more precise estimates of the political time fluctuations than a random sample would do. However, to avoid that the panel becomes obsolete or the members worn out, the panel is made rotating, i.e. it is slowly renewed. Each Friday, the n oldest members of the panel are removed while n new members are inserted in the panel. It is assumed that the organization at any point of time must have the possibility to update the information about individual panel members who have moved, changed telephone numbers, etc.

It is desired that the management of the panel members and their answers is implemented in a system which can be accessed from the net because the staff of the organization works from different locations.

Application design

The overall composition of the application design is outlined in Figure 5.1. We can distinguish between a database and a set of processes working with the database which is typical for many dynamic web applications.
Our application is implemented by a mixture of 5 HTML pages and 6 PHP scripts interacting with a database.

**Application menu**

The first and obvious task is to create an opening page with a menu by which the user can select the action wanted. A simple HTML page will provide the service needed. The following `index.htm` file is the implementation used in our example:

1. `<!--- index.htm --->`
2. `<center>`
3. `<h2><font color="Blue">Opinion polls</font></h2>`
4. `<p>The Opinion polls system is <font color="Red">initialized</font> with a sample of panel members. Each Monday a <font color="Red">List</font> of panel members to be interviewed is generated. The data for panel members can be <font color="#FF0000">updated</font> if necessary. After the interviews, the votes are recorded <font color="Red">recorded</font>. The table is the basis for computing <font color="Red">statistics</font> for the week. At the end of the week, the first panel`
member on the list is <font color="Red">deleted</font>, and a new member <font color="Red">added</font> at the end of the list.</p>

5. <table>
6. <tr><td><a href="form.htm">Initialize</a> a table of panel members</td></tr>
7. <tr><td><a href="list.php">List</a> panel members for interviews</td></tr>
8. <tr><td><a href="form3.htm">Update</a> data for panel member</td></tr>
9. <tr><td><a href="vote.htm">Record</a> interview votes</td></tr>
10. <tr><td><a href="Compute.php">Compute</a> statistics for the week</td></tr>
11. <tr><td><a href="form2.htm">Delete</a> first and add new panel member</td></tr>
12. </table>
13. </center>

The page has a simple and ordinary structure using a tags for providing the links to the 5 different services included in the system. A table tag with associated tr and td tags are used to give the page an orderly appearance.

Creating records and a list of panel members

The polling of opinions requires a sample of persons to interview. If this had been a course in sample surveys, we would have spent considerable time on how to get a representative sample of the voting population. In this course, we assume that the statisticians have completed their job, and that a list of names, etc. exists. A facility for inserting these data in the database is now needed. It is implemented by an HTML and a PHP file.

The first file is named form.htm. It includes in Line 4 a FORM tag with METHOD="post" and referring to file add.php. The METHOD="post" is important because we need it for creating global variables. The FORM block includes input boxes for Family Name, FirstName, Telephone, Age, and geographical Area.

1. <!-- form.htm -->
2. <center>
3. <h2><font color="Blue">Form to be used for adding new members to the interview panel</font></h2>
4. <form action="add.php" method="post">
5. <table>
6. <tr><td>Family name:</td><td><input type="text" name="FamilyName"></td></tr>
7. <tr><td>First name:</td><td><input type="text" name="FirstName"></td></tr>
8. <tr><td>Telephone no:</td><td><input type="text" name="Telephone"></td></tr>
9. <tr><td>Age (18-100):</td><td><input type="text" name="Age"></td></tr>
10. <tr><td>Area (1-10):</td><td><input type="text" name="Area"></td></tr>
11. <tr><td><input type="submit" name="NewMember" value="Submit new panel member"></td></tr>
12. </table>
13. </form>
14. </center>

Figure 5.2 displays the form for including a new member into the panel.
When this form has been completed and submitted to the server, the `add.php` file is called. The first action is defining and specifying the variable `$d` containing to the database named “db”. You are of course free to select any convenient name for your database. In Line 4, this variable is used to open/create the database and generate the database resource handle `$db`. Later in this script, the reference to the open database is always `$db`. The first time this script is called Line 4 will create the database.

As you recall, we created the database without specifying any tables. The first time this script is run, the required table for storing the panel member data must be created. In Lines 5-6, we consult a system table, `sqlite_master`, in which each established database and included tables are recorded. The handle `$result` contains the output from this SELECT, and Line 6 provides the number of rows of the named table, i.e. 0 if the table does not exist and 1 if established. If the table does not exist, the if-condition in Line 7 will be TRUE, and the table will be created as specified in Line 8. If the table exists, the next action is specified in Line 10 which contains the SQL INSERT statement which is the basic statement for populating database tables. Remember that because the first column `id`, an INTEGER PRIMARY KEY, is automatically generated.

```php
1. <!-- add.php -->
2. <?php
3. $d="db";
4. $db=sqlite_open($d);
```
5. $result = sqlite_query($db, "SELECT name FROM sqlite_master WHERE type = 'table' AND name = 'Voters' ");
6. $rows = sqlite_num_rows($result);
7. if($rows) {
8. sqlite_query($db,"CREATE TABLE Voters(id INTEGER PRIMARY KEY, FamilyName TEXT, FirstName TEXT, Telephone TEXT, Age TEXT, Area TEXT, Vote TEXT)" );
9. }
10. sqlite_query($db,"INSERT INTO Voters(FamilyName, FirstName, Telephone, Age, Area, Vote)
11. print("<center><font color=blue> ");
12. print("<h3>$_POST[FirstName] $_POST[FamilyName] has been added to list of voters.</h3>");
13. print("<a href=index.htm>Return</a> to menu. ");
14. print("<font></center> ");
15. ?>

Each Monday the votes from all panel members are collected by telephone interviews. As a basis for the interviews, a list of all current panel members is needed for distribution to interviewers. The list is generated from the file list.php called directly from the menu. As you can see, the content of this file is a mixture of HTML and PHP. (Note that the file extension must be .php): The title and the heading of the list is specified by HTML Lines 2 - 5 while the remaining of the file is a PHP script. This mixture design is convenient even though the same result could have obtained with 'pure' PHP only using the print() function to specify also the title, table tag and table heading.

1. <!-- list.php -->
2. <center>
3. <h2><font color=Blue>List of panel members</font></h2>
4. <table>
5. <tr><th><b>ID</b></th><th><b>Family Name</b></th><th><b>First Name</b></th><th><b>Telephone</b></th><th><b>Age</b></th><th><b>Area</b></th></tr>
6. <?php
7. $d="db";
8. $db = sqlite_open($d);
9. $r=sqlite_query($db, "SELECT * FROM Voters ORDERED BY Id ASC");
10. while ($row = sqlite_fetch_array($r)) {
12. }
13. print('</table>');
14. print("<a href=index.htm>Return</a> to menu. ");
15. print("</font></center> ");
16. print('</center>');
17. ?>

Next, the database name must be specified and SQLite opened in Line 8. In the following line, an sqlite_query[] is called with a "SELECT * FROM Voters ORDERED BY Id ASC" string. (ASC is an abbreviation for ascending). Recall that the variable Id is an automatically generated PRIMARY KEY assigned to the inserted members as integers with increasing values.
In other words, the first record in the ASC ordering has the lowest Id value and the last record is the oldest member of the panel.

The while-block in Lines 10 - 12 fetches 1 row represented as an array from the query result referred to by $r$. The array of row elements is $row$, and the individual elements of the row are $rov[0]$, $rov[1]$, $rov[2]$, $rov[3]$, $rov[4]$, and $rov[5]$ used in the print() of Line 11. The last column, Vote, in the table is excluded to avoid that the interview is influenced by previous votes.

There may be changes in name, telephone number, age, and area since last interview. To take changes into account, the list can be updated. The file form3.htm includes an HTML update form which calls on the PHP update.php file for processing the update data submitted. Do not forget that METHOD="post" is required in the form tag. Figure 5.3 shows the form for updating the individual data for a panel member.

```
1. <!-- form3.htm -->
2. <center>
3. <h2><font color="Blue">Updating data for panel member</font></h2>
4. <p>Click the list option to find the Id for the panel member, and complete the form with the updated data (all fields must be completed):</p>
5. <table>
6. <form action="update.php" method="post">
7. <tr><td>Member Id:</td><td> <input type="text" name="Id"></td></tr>
```

Figure 5.3: Updating individual data for panel members
The script of update.php demonstrates how to update (change) records of a database table. It uses the variable $_POST[Id] submitted by the form3.htm in a WHERE clause to locate the row to be updated. All other variables except Vote can be changed. The focal line is Line 5 and the SQL string surrounded by double quotes. You can forget double quotes within double quotes, but do not forget to enclose all strings to be sent to the database by single quotes. Line 5 exemplifies the main way to maintain correct values in the populated database.

Lines 6 - 11 generates an HTML page to be returned confirming that the voter with ID= $_POST[Id] has been updated. Note that this updating requires that all boxes of form3.htm must be completed, even if not all data have changed. This drawback can easily be eliminated by making form3.htm to an .php vile, and including an sqlite_query() with an SQL SELECT statement for the particular id. This would have given us the current data with which we could pre-fill the form and we would only have to type in the values which have changed.

Processing, statistics and rotation

After the panel members have been interviewed, their votes must be recorded in the system. A simple HTML page of the following type can be used:
2. <center>
3. <h2 style="color:#0000FF">Vote recording</h2>
4. <p>Print out the list of panel members. Use the Id number from the list when recording the vote of the individual interviewed person. </p>
5. <form action=record.php method=post>
6. <table>
7. <tr><td>Id number of panel member</td><td><input type="text" name="id" size=4></td></tr>
8. <tr><td>Vote</td><td><input type= text name="vote"></td></tr>
9. <tr><td></td><td><input type=submit value=Record></td></tr>
10. </table>
11. </form>
12. <a href="index.htm">Return</a> to menu.</p>
13. </center>

Figure 5.4 shows the form for recording the votes.

![Vote recording form](image)

Figure 5.4: Recording a vote

To preserve privacy as much as possible, only Id number and Vote are recorded. The form tag specifies the record.php as the script for processing the submitted data.

This script demonstrates an sqlite_query() function with an SQL UPDATE string by which the record for the specified id is updated with the vote given in Line 6.
One of the options in the menu is to compute the statistics for the week. This option calls the `compute.php` script directly. The statistics computed are quite simple and meant only to be an example. It starts by defining 5 alternative votes, $A$, $B$, $C$, $D$ and $E$ and setting these initially equal "0" in Line 3 - 7.

```php
<!DOCTYPE html><html><head><title>Vote Results</title></head><body>
<?php
$A="0";
$B="0";
$C="0";
$D="0";
$E="0";
$d="db";
$db=sqlite_open($d);
$r=sqlite_query($db,"SELECT Vote FROM Voters");
while($row=sqlite_fetch_array($r)) {
  $s=$row[0];
  switch($s) {
    case "A": $A++; break;
    case "B": $B++; break;
    case "C": $C++; break;
    case "D": $D++; break;
    case "E": $E++; break;
  }
}
print("<center><font color=blue><h2>Vote frequencies this week</h2></font>");
print("<table>");
print("<tr><th>Alternative:</th><th>Frequency:</th></tr>");
print("<tr><td>A</td><td>$A</td></tr>");
print("<tr><td>B</td><td>$B</td></tr>");
print("<tr><td>C</td><td>$C</td></tr>");
print("<tr><td>D</td><td>$D</td></tr>");
print("<tr><td>E</td><td>$E</td></tr>");
print("</table>");
print("<a href=index.htm>Return</a> to menu.");
?>
</body></html>
```
In Line 10 all values in column Vote of table Voters are retrieved, and the script continue with a counting loop in Line 11 - 24. For each new row the vote (in $row(0)) is assigned to the variable $s. The multi-branch switch($s) statement is used in the head of a loop in Line 13 - 23., and depending on the case of the current row, the corresponding cumulating variable is incremented by 1 (obtained by the operator ++). When all rows have been traversed, the cumulated values of the variables are printed out.

More sophisticated and useful statistics can be produced if we define the database table to take care of both current and previous week's votes. Such statistics can tell us about the voters migration from one party to another during the week.

At the end of each week, the panel must be rotated, i.e. the n first (longest serving) members should be deleted and replaced by n new members. Without loss of much generality, we have simplified the rotating to delete the first member only and add one new member. The form2.htm page, shown in Figure 5.5, takes care of the recording of the new members' personal data, and calls the delete.php script at the server to complete the task.

![Delete first and add new member to the panel](image)

Figure 5.5: Rotating the panel of interview members

1. <!-- form2.htm -->
2. <center>
3. <h2><font color="Blue">Delete first and add new member to the panel</font></h2>
4. <h3>New member data:</h3>
rotate.php retrieves all records from table Voters and order them by ascending value of the attribute Id. Since Id is a PRIMARY KEY automatically assigned, a newer record will always have a higher id value than an older. The first array obtained by sql_fetch_array() in Line 6 will therefore be the member with the longest service at the panel and the one to be deleted. Line 7 contains the SQL statement DELETE used for removing records from a populated table. In Line 8 the new member is inserted in the panel.

Improvements to this application have been suggested at several places in the text. You can for example try to introduce gender, i.e. separate categories for mail and female voters, and/or expand the database table to take care of 2 weeks' votes for each panel member. In the computation script, this will permit comparison of special tables for male and female voters. The computation script can also be extended to generate tables displaying the migration of voters among parties from last to current week.
Session 6: Functions in PHP

Functions

One of the most frequently ways of re-using code is creating a function. A function can be considered as a complex operator, and is either a built-in function or a user-defined function. A function is recognized in the script by a name followed by a pair of parentheses which embraces none, one or several arguments. While the names of variables are case-sensitive in PHP, the function names are not case-sensitive.

We have already met a number of built-in functions such as print(), sqlite_query(), etc. The internal functions come with the language processor and are available for call when needed.

The user-defined functions must, as the name indicates, be defined and be made available by the developer before they can be called from a script. The syntax for defining a function is:

```
1. function my_function ($a, $b, $c) {
2.    function code
3.    return $d
4. }
```

The specified arguments $a, $b, and $c (your definition determines the number and names of the arguments) are called the formal arguments and are the names you use in the function definition. If the function returns any values, they are represented by $d which can be a single variable or an array. A defined function can be called from a script if it is accessible for the script. It can be made available either by copying the complete function definition into the script, or by inserting an include ("my_functions.php") in the beginning of the calling script file. The advantages of the latter option are saved space, and that the file my_functions.php can contain several custom-made functions needed to be called from several script.

A call from a script to a user-defined function is by the statement $D=my_function($A, $B, $C). The argument names $A, $B, and $C, called the actual arguments, are the variable names used in the calling script for the variable values you want to pass to the function. It is important to note that these values are given to the formal variables $a, $b and $c within the function while the variables $A, $B and $C outside the function maintain their values. The function value $D is the output return from the function available to the calling script from the return variable $d in the definition. In some cases, a function may have no output and this variable and the assignment operator are not needed. The function can also return several values in which case $D is an array.

Note that the symbols used here for formal and actual variables are examples. You are free to use the names of your own choice in your functions and applications.
**Authorization and authentication**

By **authorization** we mean the assignment of access identities to a web site visitor. **Authentication** is the checking of the validity of identities provided by a visitor to obtain access to the site. When designing a web application, authorization and authentication is frequently required functionalities for varying reasons. The site owner may for example want to know who are visiting the site, to be able to provide **personalized service** to customers, to keep track of the **performance** of student visitors, etc.

These functionalities are used with minor variation in different applications. As an example of more user-defined functions, we will design a **login** module which can perform authorization and authentication in web applications. **Figure 6.1** indicates the overall structure of the login functionality. There are alternative designs which could have been used. If for example it is important to preserve the anonymity of the users, we could have asked the user to select a **PIN** code herself and let the server **hash** the selected code and check that it is **free**. The **PIN** external code would then be **unavailable** for the host while the anonymous internal code could be used for analysis of the site visits. However, a hash code approach would not have satisfied applications in which it is required that the host can recognize the users' external identities, e.g. in e-shop billing and e-courses with grading.
We start the example application by developing 2 ordinary HTML pages. The first, `index.htm`, generates the login form returned to the visitor when calling the application.

1. `<!-- index.cfm -->`
2. `<center>`
3. `<h1><font color="Blue"> <font size="+3">Login</font></font></h1>`
4. `</center>`
5. Thank you for your interest in this site. To get access to the application, you have to be authorized.<i>If you already are registered</i>, please go directly to the login.<br>
6. `<i>If you are new and want access to the application</i>, we need some information from you, and you will need a personal identity number (PIN). Please continue with the `<a href="registration.htm">registration</a>`.
7. `<p></p>`
8. `<center>`
9. `<table>`
10. `<tr><td><font color="Red"><b>Login with your</b></font></td></tr>`
11. `<FORM ACTION="functioncalls.php" method="post">`
12. `<tr><td>Your username:</td> <td><input name="username" type="text" size="20"></td> </tr>`
13. `<tr><td>Your PIN code:</td><td> <INPUT TYPE="password" name ="submitted_pin" SIZE="20"></td></tr>`
14. `<input name="login" type="hidden" value="1">`
15. `<tr><td>Click the button:</td> <td><INPUT TYPE="SUBMIT" NAME="response" VALUE="Submit"></td></tr>`
16. `</FORM>`
17. `</table>`
18. `</center>`

This form provides the server with the variables `username` and `PIN` used for authentication. The only new aspect in this page is the use of the attribute HIDDEN in the INPUT tag in Line 14. The hidden variable named `login` with `value="1"` is invisible for the client. By means of this variable, the server will be able to distinguish the variables sent by this page from those sent from the next form. See Figure 6.2. The form calls the script `functioncalls.php` which represents an application for which we require controlled login.
The form also includes a link to registration.htm for visitors who are not yet registered. The form in this second HTML page collects the information required for authorization, in this example first and last name and a username chosen by the visitor. Other personal information such as gender, age and home region, can of course be included if required.

```
 1. <!-- registration.htm -->
 2. <html>
 3. <head>
 4. <title>applications</title>
 5. </head>
 6. <center>
 7. <h1><font color="blue"><b>Registration</b></font></h1>
 8. <p>In order to recognize and serve the different requirements of our visitors, each visitor needs her/his own username and PIN code.<br> Please, complete and submit the form, and your username and PIN code will be returned to you.</p>
 9. <FORM ACTION="functioncalls.php" method="post">
10. <table>
11. <tr><td>Your first name:</td> <td><input name="firstname" type="text" SIZE="20"></td> </tr>
12. <tr><td>Your last name:</td> <td><input name="lastname" type="text" SIZE="20"></td> </tr>
13. <tr><td>Your user name:</td><td> <INPUT TYPE="text" name ="username" SIZE="20"></td></tr>
14. <input name="registration" type="hidden" value="1">
15. <tr><td>Click the button:</td> <td><INPUT TYPE="SUBMIT" NAME="response" VALUE="Submit"></td></tr>
```
Note that also this page has a hidden input variable, named registration with value="1", for the same reason as the first form. This form is shown in Figure 6.3. This form specify the same PHP script, functioncalls.php, for server processing as did the first form.

![Registration](image)

Figure 6.3: Registration form for new visitors

The PHP script functioncalls.php can be considered as the application script of the example. It contains several new features. Line 3 includes another file, in this case functions.php. The file functions.php contains 4 functions we shall discuss in detail below.

The first function called is sql_table_exists() in Line 7. We know the functionality from the previous session. Below we shall put the functionality into a formal function. If the table Users in the database $db does not exist, it is created in Line 8. The purpose of this table is to keep track of all authorized users and their identities.
1. <!-- functioncalls.php -->
2. <?php
3. include "my_functions.php";
4. //Open database
5. $d= "db" ;
6. $db=sqlite_open($d);
7. if(!sqlite_table_exists($db, 'Users')) {
8.   sqlite_query($db, "CREATE TABLE Users (firstname VARCHAR(20), lastname VARCHAR(20), username VARCHAR(20), PIN VARCHAR(10))");
9. }
10. if (isset($_POST['login'])){ 
11.   $approved=authentication($db, $_POST['username'], $_POST['submitted_pin']);
12.   if ($approved[0]=="yes")
13.     print("<h2><center><font color=blue>$approved[1], you are logged in</font></center></h2>");
14.   else
15.     print("<p><center><font color=red>Your PIN code was invalid</font></center></p>");
16. }
17. if(isset($_POST['registration'])){ 
18.   $reg=authorization($db,$_POST['firstname'],$_POST['lastname'],$_POST['username']);
19.   print("<center><font color=blue>You have been successfully authorized to access the site<br> Your username is: $reg[0], and your PIN is: $reg[1].</center></p>");
20.   print("<a href=index.htm>Return to Login</a>.");
21. }
22. ?></p>

In Line 10 the existence of the hidden variable login is tested by means of the built-in function isset(). If it has been assigned a value, the server will know that it is a login form it has received, and a function call to the function authentication() is made in the following line. The authentication function requires 3 arguments, the database handle $db, as well as the $username and the $PIN submitted by the user on the login form. A return value for the variable approved is expected from the function. If the returned value is "yes", a message is sent back to the client in Line 14 confirming that she is logged in. If not, the login failed, and a message about the failure is sent by the next couple of lines.

If a value for the hidden variable registration is received from the client instead of login, the function authorization() is called. This function is expected to return the values of username and PIN. The function output must therefore include 2 variables, in this case the username and the PIN in an array with 2 elements defined within the function.

Table existence test

In the previous session, we studied how to test for the existence of a table. This is a task frequently met, and it is well suited for implementation as a function.

1. //Table existence test
2. function sqlite_table_exists($sqlite, $table)
3. $result = $sqlite->query("SELECT name FROM sqlite_master WHERE type='table' AND name='$table'");
4. return $result->numRows() > 0;
5. }

The function should in our context be called by the statement `sqlite_table_exists($db,'Users').`

**Authorization**

We proceed to the definition of the functions, and start with the definition of `authorization()`.
This function should be able to generate a new, unused PIN, append the row with the persons first name, last name, user name and generated PIN in the created table of the database.

The task of generating a new PIN code is left to another function, `pin_assignment()` which only require the database handle, $db, as argument. This function, which is the next to be discussed, delivers a new and unused $PIN.

The data for the new visitor is inserted into the database table Users by a `sqlite_array_query()` function which is well suited for this purpose. We want the function to return 2 variable values, $username and $PIN, in an array which is obtained by the return statement in Line 5.

**PIN code assignment**

We saw in the previous function definition that one function can call another. As a matter of fact, a function can call itself which results in a recursion. In the `pin_assignment()` function, the first step is to retrieve all records(arrays) from database table Users in Line 3, and set initially the test variable $used="yes" before entering the random number generation and testing while loop in Line 5.

In Line 6 a random seed is planted followed by generation of a random integer, $PIN2, in the range 1000 to 99999 (these limits can be set according to the particular needs). The random seed is planted to avoid that the same number is generated each time the application is run.

The variable $used is now re-set to "no". A test to see if the generated number has been used before, runs from Line 9 to 13. Here $result is the array corresponding to the current row $r of the database table and $result['PIN'] is the value of the PIN column of that row. If this $result['PIN'] is identical (==) to the generated $PIN2, the generated integer is used, and variable $used is assigned the value "yes". There is no reason to continue the test, the loop is
therefore broken by the break statement in Line 12, and the processing is directed back to Line 5.

A new integer is generated and the test repeated, until an integer not used in the table Users is found, i.e. the test loop is exited with $used="no". The final statement is to return the unused integer $PIN2.

**Authentication**

The third function we need is the authentication(). The call passes $db, submitted $username and $PIN code to the function. The task for this function is to check if the passed values are valid according to the User table. To avoid problems with disturbing white space, etc. in connection with the string values, the sqlite_escape_string() function is used in Line 3 and 4. 2 arrays are defined. in the next lines. $result will be used for storing the retrieved rows (usually 0 or 1 row) from table Users, while the array $approved will be used to return 2 values to the calling script.
14. else $approved[0]="no";
15. return $approved;
16. }

After possible retrieval of rows where the PIN column has the value $PIN2, the content is inserted into the array $result in Line 8. In the next row, we use the concatenation operator (".") and 2 concatenated strings, $result['username'].$result['PIN'] and $username2.$PIN2, are compared. By concatenating username and PIN code, we have the extra security that the combination of username and PIN code are validated.

If the submitted data are validated, the first element $approved[0] of the array $approved is assigned the value "yes", and the second element $approved[1] is assigned the value $result['firstname'] from the matching row of the database. If no the first element is set to "no". The last action is to return the array $approved to the application script.

Function library

The 4 functions created for this application can either be copied into the PHP files in which they are called, or, as we have done in this example, simply be collecting all functions in a library file called my_functions.php. The structure of this file is:

1. //Namer: My_functions.php
2. //Function:sqlite_table_exists is copied here
3. //Function: authorization is copied here
4. //Function: pin_assignment is copied here
5. //Function: authentication is copied here

The library file is stored in the same directory as the other files of the example. The functions are made available to the application with the include() statement as demonstrated in the script functioncalls.php.

More function can be added into the file which in fact becomes a library file and can be used in different applications which need one or more of the functions. The advantages are that all functions are kept in one location making maintenance more effective.

Logging

To demonstrate the last statement, and to introduce a few more aspect, we introduce a second example, the logging function. By logging we mean recording when identified users are passing specified observation points in our application system. Logging is particularly important for the
application usage analysis when studying how a system is actually used and for preparing improvements based on this experience.

**Logging function**

Below is a function definition, `logging.php`, which require a single argument, a PIN identification. The value of the current user's PIN is frequently available as `$_SESSION['PIN']`, and can be used as argument in calling this function.

```php
//Logging.php
function logging($PIN) {
    if(!file_exists('/PHPRoot/log.htm')) {
        $f=fopen('/PHPRoot/log.htm','wb');
        fwrite($f, "LOG FILE<br>");
        fclose($f);
    }
    $time=strftime('%c');
    $record="$time, $PIN, $_SERVER[SCRIPT_FILENAME]<br>";
    $stream='/PHPRoot/log.htm';
    $f=fopen($stream,'aw');
    fwrite($f, $record);
    fclose($f);
}
```

The function specification, `logging.php`, is stored as the fifth function in the `my_functions.php` file.

**Example environment**

To be able to demonstrate the use of this function, we shall need a few more surrounding files: a menu `index.htm`, `start.php`, `view.php`.

The `index.htm` looks like this:

```html
<!-- index.htm -->
<html>
<head>
<title>Index.htm</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<center>
<h2><font color="blue">Logging menu</font></h2>
Do you want to:
<table>
<tr><td>Make a log entrance:</td><td><a href="start.php">Yes</a></td></tr>
<tr><td>See the log:</td> <td><a href="view.php">Yes</a> </td></tr>
```
The only justification of this page is to demonstrate either initiating a log record or seeing the log file.

The purpose of the **start.php** script is to simulate the entering into an application. The first statement, **Line 2**, of this script include our function library **my_functions.php** which makes the function **logging()** available. In a real application a user will have logged in with a PIN code. In our example, we assume that the in PIN code is "1234". When our activation of the **start.php** has been logged, we will get a message in return. In a real application there are usually no need for returning messages each time a logging has been carried out.

```php
<?php
include('my_functions.php');
$PIN='12345';
logging($PIN);
print("<center><font color=blue>Your visit to this test page Index.php has been logged</font><center>");
?>
```

To complete our examplification of the logging function, we also need a short script we can call (the second option of the **index.htm**) to view the log file. **view.htm** serves this purpose. In this script, we make use of `$stream` and the powerful **file_get_contents()** function. This function opens, reads the content of and closes the specified file. The contents of the file is read into a string variable (in the example called `$contents`).

```php
//view.php
$stream='log.htm';
$content=file_get_contents($stream);
print("$contents");
?>
```

Try to identify scripts in your own application work which you think can be used in other applications and start building your private library of user-defined functions.
Session 7: File processing

So far, we have considered databases as the main storage for sets of data. However, it is frequently needed and efficient to work with serial data stored as files if random access to elements of the set is not predominant. Typical examples are the logging example of the last session, and text files where required access to the individual elements of the file usually is sequential.

In this session, we shall discuss 3 typical file applications, i.e. maintaining files on the server, fetching remote files from another server to our server, and uploading files from a client computer to our server. (Downloading files available at the server is no problem with software usually available at the client computer).

Maintaining files

In the logging example of the previous session, it was demonstrated how a file, the log.htm file, can be appended. The same technique can be used to read a file into a text string, parse and insert/change/delete a certain specified word(s) and return the maintained file to the client computer.

Fetching files

In some applications, it is required that the server downloads/uploads files from/to other servers in order to provide the intended services to its clients. As an example, consider a server which is maintaining a copy of a continuously updated news source based on a regular scanning of source.

The application example uses a continuously running Internet agent to maintain a server file with a recent copy from a news server. The demonstration of this example requires 3 programs, an index.htm page for displaying 2 example options, an agent.php script which is downloading and saving the news page every 1800 second, and a serve.php script to serve the client with the most recent news copy.

The index.htm is by now trivial and its function is only to provide the user with the option of 2 alternative requests:

1. <!-- index.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
3. <html>
4. <head>
5. <title>Untitled Document</title>
The option 1 calls the agent.php to start and run the agent. Line 5, containing a HTML meta tag, instructs the server to repeat the request every 1800 seconds as long as the message in Line 13 is visible. You can see the selection displayed in Figure 7.1. Note that this agent is running only when the window is kept open.

Figure 7.1: Menu for the agent example
The content of the news source (edition.cnn.com/index.html in the current example) is obtained by means of the stream function, file_get_contents(), which can as well be used for obtaining contents from remote files (as in the example) as well as from local files as in the previous session. Line 11 fetches the content of the specified remote file and assign it to the variable $content. The next line includes another stream function, file_put_contents(), which stores the content of a variable in a specified file. The syntax for specifying a local stream includes 2 components, file:// is called the scheme, while /PHPRoot/news.htm is called the target. Note the 3 slashes, ///. In our context, PHPRoot is the name of the top document directory of our web domain. The scheme http:// indicates that the file operation concerns a remote file in contrast to a remote file. The scheme file:// syntax indicates a local file operation.

```php
<!--- agent.php --->
<?php
file_put_contents("file:///PHPRoot/news.htm", $content);
print("The agent is running");
?>
```

As long as the agent is running, the file news.htm is updated regularly every 1800 second. To be able to keep the agent running and at the same time continue to investigate this example, you must therefore open another session for the remaining part of the example. In a real situation, the start and running of the agent will be the privilege of the web-master and the clients will be limited to requesting the latest copy of news.htm.

Each half hour (1800 seconds), the content of the file news.htm is refreshed. The users can request a copy by option 2 in index.htm which calls serve.php:

```php
<!-- serve.php -->
<?php
$news= file_get_contents('file:///PHPRoot/news.htm');
print("$news");
?>
```

The variable $news is used for sending a reply to the requesting client by means of the print() in Line 4. Note that in this script, the same stream function file_get_contents() as in agent.php (where it was used to fetch a remote file) is here used for a local target with the scheme file//.
Uploading files

So far, we have been studying file processing on the server level and downloading files to the clients. In some applications, it is required that files can be uploaded from the clients to the server. The security risk connected to this feature should not be underestimated.

As an example, consider this online course. The reports to assignments given in the course should be uploaded to a specified directory on the server. The course instructor and the students should at any time be able to see the list of uploaded reports, and if wanted, read any of the uploaded files.

index.htm presents 2 options, see the list of uploaded files or upload a report file, as displayed in Figure 7.2.
The first option of the index.htm file is to get a listing of the uploaded files from the directory (..\UploadFiles\file relative to the script) in which the saved files are stored:

1.  <!-- list.php -->
2.  <?php
3.  print("<center>");
4.  print("<h2><font color=Blue>List of uploaded files :</font></h2>");
5.  $dir="../UploadFiles/file";
6.  if ($handle= opendir($dir)){
7.  print("<b>Files:</b><br>"
8.  print("<table>");
9.  while (($file = readdir($handle)) !=false) {
10.  if ($file !=".." && $file != ".") {
11.  print("<tr><td><a href="../UploadFiles/file/$file> $file </a></td></tr>"");
12.  }
13.  }
14.  print("</table>");
15.  closedir($handle);
16.  }
17.  else print("No files uploaded.");
18.  print("</center>");
19.  ?>

If the directory exists, it is opened with a returned handle, $handle, and a list of the files stored in the directory is displayed with an HTML a tag for each file which permits the content of the individual files displayed. An example of the resulting list is shown in Figure 7.3.
List of uploaded files:

Files:

- myFile.htm
- yourFile.htm

The second option results in a second HTML page, `upload.htm`. A few special aspects of this form page should be noted. **First**, the form tag must as usual have `POST` specified as method. **Second**, the tag must include the attribute `enctype="multipart/form-data"`, and, **third**, an input tag with `attribute type="file"` must be present.

```
1. <!-- upload.htm -->
2. <center>
3. <h2><font color="Blue">Uploading</font></h2>
4. <form action="upload2.php" method="POST" enctype="multipart/form-data">
5. <table>
6. <tr><td>Name to be assigned to file at server:</td> <td><input name="name" type="text"></td></tr>
7. <input name="MAX_FILE_SIZE" type="hidden" value="30000">
8. <tr><td>Identify the file at your computer to be uploaded:</td>
9. <td><input type="file" name="upload" ></td></tr>
10. <td></td><td><input type="submit" Value="Upload file"></td></tr>
11. </table>
12. </form>
13. </center>
```

The form is displayed in **Figure 7.4**
The form page upload.htm calls the PHP script upload2.php which confirms the uploading of the file by means of the statements in Line 3-8, and stores the file by means of Line 9-10. The move_uploaded_file() is another stream function.

1. <!-- upload2.php -->
2. <?php
3. $destination_file="./file/$_POST[name]";
4. move_uploaded_file($_FILES['upload']['tmp_name'], $destination_file);
5. print("A file with the following attributes has been uploaded: <br>");
6. print(" Remote name:".$_FILES['upload']['name']."<br>");
7. print(" File type:".$_FILES['upload']['type']."<br>");
8. print(" Size in bytes:".$_FILES['upload']['size']."<br>");
9. print(" Temporary name:".$_FILES['upload']['tmp_name']."<br>");
10. print(" Error code:".$_FILES['upload']['error']."<br>");
11. ?>

Note that the concatenation operator '.' is used in the print() functions in Lines 6-9 to join the texts surrounded by double quotes with PHP array variables. You can see a confirming message in Figure 7.5.
A file with the following attributes has been uploaded:
Remote name: home.htm
File type: text/plain
Size in bytes: 129
Temporary name: C:\WINNT\TEMP\php17.tmp
Error code: 0

Figure 7.5: Server's confirming of a received file.
Session 8: Object-Orientation in PHP

Object-oriented interface

So far, we have introduced and discussed the procedural interface to PHP. In the last versions of PHP, and particularly in PHP 5, an alternative object-oriented interface has become available. The users can now choose between scripting by means of the procedural approach or using the object-oriented approach according to personal training, taste and preference.

Even if you are convinced that the procedural interface is what you are going to use, it can be useful to have certain knowledge about what you gain, or loose, by making that choice. The following section is a half-an-hour introduction to object-orientation followed by 2 sections in which application of this interface to PHP 5 are discussed and demonstrated.

Object-oriented syntax

The idea of object-oriented programming was conceived in 1964 and implemented in the programming language SIMULA. The basic ideas were the concept class and data abstraction from which objects were derived. An object in PHP is thus a set of methods (functions) which can operate on an embedded set of properties (data). An object can be invoked (called) to perform its specific task from outside.

When modeling an application, it is usual to divide the application into part which often correspond to real life objects, e.g. teacher, students, login, information, sessions, tests, assignments and exercises. Each of these can be a class candidate. A class defines the typical structure of a set of objects and is used to specify objects for the application. Like functions, there are classes included in PHP and classes defined by the users.

Important properties of object-oriented scripting are encapsulation and inheritance.

Classes and objects

A class can be considered as a template defining a structure of properties and methods from which individual objects can be derived. The class student is composed by:

```plaintext
1. class student {
2.   properties
3.   methods
4. }
```

The class definition must be available for a script to instantiate one or more objects from the class. The most primitive option for making the class available is to copy the class definition within the script. A more practical solution is, however, to collect class definitions in a class library file, for example:
When a class, e.g. called student, is defined and available, objects can be instantiated by the statement:

$john= new student;

and the object $john can be used in applications. As we shall see later more information can be provided in the new statement to create more powerful objects.

**Properties and methods**

Properties are data used within the object. The class student can be defined to include the variables #studentname, and #coursename. If they should only be available to the methods within the object, they are declared private. They may also be declared public, but declaring public variables within a class are not considered good practice if not required for for data transfer to and from the objects.

The class student can now be specified as:

```php
class student {
    private $studentname;
    private $coursename;
    methods
}
```

Since the 2 variables are declared private, i.e. cannot be used outside the object, the next thing needed is a method to give access to the object to assign values to the variables. Such a method can be defined by extending the class definition with the specification of a public function (note that in OO programming the methods are implemented as functions) with 2 formal arguments, $name1 and $name2. By means of this function we shall later be able to set names in the object:

```php
class student {
    private $studentname;
    private $coursename;
    public function setname($name1,$name2){
        $this->studentname=$name1;
        $this->coursename=$name2;
    }
}
```

The class definition contains a couple of new syntactical symbols, the variable $this and the symbol ->. The variable $this refers to the class in which the function is defined and -> points to an variable in the class which should be assigned the value on the right side of the assignment operator. Note that variables after the symbol -> do not start with $.
Let us complete the class definition with a second method, public function getnames(), which will enable us to retrieve the names of the student and the course. The definitions of the public methods are included in the complete class definition below:

```php
<?php
class student {
    private $studentname;
    private $coursename;
    public function setname($name1, $name2) {
        $this->studentname=$name1;
        $this->coursename=$name2;
    }
    public function getnames() {
        $names['studentname']=$this->studentname;
        $names['coursename']=$this->coursename;
        return $names;
    }
}
?>
```

In the second method, the studentname and coursename are copied into an array called $names which is the return variable of the getnames(). The class is saved in a PHP file named myClassLibrary.php in directory /PHPRoot/ for easy access. In this file and within the PHP script tags, more class definitions can be stored.

Let us illustrate the use of the class in a mini-application consisting of the following HTML page and PHP script. The HTML page is a simple form page called index.htm and illustrated in Figure 8.1:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>

Class application

This example is a first demonstration of object-oriented programming in PHP 5. It contains a single class called student. Objects instantiated from this class can read to names take care of the names and retrieve them on request. Type a student name and a course name:

Student name: ____________________________
Course name: ____________________________

Sand

Figure 8.1: A class application form
```
This example is a first demonstration of object-oriented programming in PHP 5. It contains a single class called student. Objects instantiated from this class can read to names take care of the names and retrieve them on request. Type a student name and a course name:

```
<form action="main.php" method="post">
<table>
<tr><td>Student name:</td><td><input name="studentname" type="text"></td></tr>
<tr><td>Course name:</td><td><input name="coursename" type="text"></td></tr>
<tr><td></td><td><input name="" type="submit" value="Send"></td></tr>
</table>
</form>
```

Note that the form tag refers to a script file called `main.php`. The HTML form page should by now be well known and not require any further explanation.

The PHP script is short, but can deserve a few comments.

```
<?php
include "/PHPRoot/myClassLibrary.php";
$my_student= new student;
$my_student->setname($_POST["studentname"],$_POST["coursename"]);
print("The following names are processed by object my_student:<br>");
$namses= $my_student->getnames();
print("Student name: $names[studentname]<br>");
print("Course name: $names[coursename]<br>");
unset($my_student);
?>
```

It is usual that an object-oriented application has a 'main' file by which the messages to and from the objects are initiated and received. Line 3 makes the class definition(s) available by an include statement. Next one or more objects must be created. In our example the object is called `$my_student` and created in Line 4. A method in the object is called by a statement starting with the name of the object followed by the symbol -> and the name of the function with a pair of parentheses which may or may not enclose arguments. In this example `$my_student->setname($_POST["studentname"],$_POST["coursename"]`) In Line 7, the second method is called returning the array `$names`. From the following 2 print(), we see that the `$names` is an array. Remember that because this array is located within a pair of double quotes in the print functions, no single quotes must surround the names of the array's keys. Finally, the object is closed by the function `unset()` in Line 10. The output of the object $me is shown in Figure 8.2.
Frequently, a class definition will also include a couple of special methods, constructors and destructors, the purpose of which is to prepare the object for use when invoked and to destroy an object when it is not needed any more. A constructor function is not called, but automatically executed when a new object is invoked. The example has neither constructor nor destructor.

SQLite OO interface syntax

A number of classes are built into the object-oriented interface of the PHP language. Let us return to the application of SQLite to study some of the built in classes. With the OO interface, a database object is invoked by using the SQLiteDatabase class. The syntax is:

```
$db = new SQLiteDatabase('mydatabase.db');
```

The class SQLiteDatabase has obviously a constructor function setting the value of the class property value 'mydatabase.db'. Our database object can be queried by:

```
$r = $db->query($sql);
```
where $sql can be a SQL statement, e.g. 'SELECT * FROM users'. Note that this call does not require any reference to $db within the query because the call itself is made to a method in object $db.

The query() method is most useful when the result is a small number of rows. Another method, unbufferedQuery(), should be used when the number of rows exceeds 50:

$r=$db->unbufferedQuery($sql);

Other useful methods available in the object $db are:

$r=$db->arrayQuery($sql);

and

$secure=$db->escapeString($sql);

which are obvious counterparts to SQLite functions in the procedural interface.

A useful feature of PHP 5 is iteration which permits access to rows from a database query as though they were elements in an array. The while loop is an example of iteration. Another iteration is foreach used to process the rows from a query:

$r=$db->query($sql);
foreach ($r as $row) {
  // one at a time each $row becomes available for processing
}

Database error handling is possible in the OO interface by using the try and catch. If an error occur in creating an object, the new statement can be tried, and any error caught as demonstrated by the following tryCatch example.

In an HTML form page, the user can choose between a blank or 'database' as name for the database file.

1. <!DOCTYPE HTML PUBLIC "-/W3C//DTD HTML 4.01 Transitional//EN">
2. <html>
3. <head>
4. <title>Untitled Document</title>
5. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6. </head>
7. <body>
8. <center>
9. <h3><font color="blue">Try and Catch</font></h3>
10. <p>Choose one of the 2 alternative names for a database to be created/opened, and observe what the Try and Catch reports.</p>
11. <table>
12. <form action="tryCatch.php" method="post">
The control is left to the script `tryCatch.php`, and in its Line 3, creation of a new database object is tried. If a valid name is provided by means of the form page, e.g. 'database', the script returns in Line 9 a message that the operations was successful. If an error is caught in establishing the database object, an error object is activated and by means of 2 methods, `getMessage()` and `get_file()`, the user is informed about the error(s) and which files caused the error(s).

```php
<?php
    $d=$_POST['name'];
    try {
        $db= new SQLiteDatabase($d);
    } catch(SQLiteException $error) {
        print("Error message: " . $error->getMessage() . "<b>");
        print("Error caused by:" . $error->getFile() . "<br>");
        die;
    }
    print("The named database was successfully created/opened.");
?>
```

If the name of the database is blank in Line 2, running the script gives the following message:

```
Error message: SQLiteDatabase::__construct() [function.--construct]: unable to open database: C:\PHPRoot\Examples\Session8\tryCatch
Error caused by:C:\PHPRoot\Examples\Session8\tryCatch\tryCatch.php .
```

Note the `die()` function in Line 7 which causes the script stop. Exceptions are only thrown in the object-oriented interface in PHP 5, not in the procedural interface.
Session 9: Login class

Authentication and authorization re-visited

To continue the introduction of the object-oriented interface to PHP 5, we return to the example of authorization and authentication for which we will develop a class Login. The objects of this class should be capable to register a new uses with her first and last names, as well as her email address as user name, check if the email address has already been used as user name and if so reject the registration. For the case the email address has not been used, the objects should include a method which generates a unique 5 digit PIN code and return a message containing the email address and the PIN code. An object of this class should also be capable to receive an email address and a PIN code and check that their concatenation exists for an authorized user.

Implementation of an example demonstrating the use of such a class requires a couple of HTML pages for providing the application with the necessary user data, a PHP script representing some application invoking an object of the class and passing the user data to the object and processing the returned messages from the object, and finally access to the class Login.

HTML form pages

The HTML form pages are quite similar to form pages already discussed in connection with the example A&A in session 6, but observe that there are certain differences!. The first, index.htm, the login page, also contains a link to the second for visitors not already registered or for other reasons without a valid username and password.

```html
1. <html>
2. <head>
3. <title>index.htm</title>
4. </head>
5. <body>
6. <center>
7. <h1><font color="Blue">Login</font></h1>
8. </center>
9. Access to this site is restricted to registered visitors only.<i>If you already are registered</i>, please go directly to the login.<br>
10. <i>If you are new and want to become a registered user</i>, we need some information from you, and you will need a personal identity number (PIN). Please continue with the <a href="registration.htm">registration</a>.
11. <p></p>
12. <table>
13. <tr><td><font color="Red">Login with your</font></td><td><input name="email" type="text" size="20"></td></tr>
14. <form action="application.php" method="post">
15. <tr><td>Your username:</td><td><input name="email" type="text" size="20"></td></tr>
```
For registered users, the form requires that the visitor types his **username** (=email address) and **PIN** code, and the control is transferred to a script, `application.php`. This script connects any application using the login object. To be able to coordinate this page with the registration.htm discussed below, **2 hidden input blank values** for **firstname** and **lastname** are required. The reason will become obvious when we discuss the **class** below. Note the differences compared with the one used in the session on user-defined functions.

The **registration.htm** form page refers to the same script, `application.php`, as the **index.htm**. It requires first name, last name and email. The email is later used as username.

Note the hidden input of a **blank pin** value in this page.
Both the 2 form pages discussed refer to application.php (the name is unessential and can easily be adjusted to fit the application situation. The purpose of this script is to connect the pages to the login object and the login object to the application.

```php
<?php
//application.php
$continue="www.site.htm";
/* Except for the above line, which must be customized, the remaining script should not be changed*/
include "//PHPRoot/myClassLibrary.php";
$myLogin= new login;
$result=$myLogin->main($_POST['firstname'],$_POST['lastname'],$_POST['email'],$_POST['pin']);
if ($result[0]="yes") {
print("<center><font color=blue>$result[1] $result[2], you have been logged in<br>Please <a href=$continue>Continue</a></font></center>");
}
if ($result[0]="no") {
print("<center><font color=red>Your email/username or pin was rejected</font></center>");
}
if($result[0]="new") {
print("<center><font color=blue>Your username is: $result[1]<br>Your password is: $result[2]</font></center>");
}
unset($myLogin);
?>
```

In Line 3, a variable $continue is introduced to link to the specific application you are logging into. The URL of the application should substitute the www.website.htm. To be able to create an object from the class Login, the class must be made available to the script. We imagine that the class is saved in a .php file named myClassLibrary.php located in a directory in C:/PHPRoot and use an include() in Line 5 to make the class available (if you choose another directory, you must adjust the path). An instance, $myLogin, of the class is created by Line 6. The call for the object in the form is in Line 7 in which the public function in the object is called by $myLogin->main(). It is important to notice that the variable marker, $, only occur before the name of the object, and not in front of the function name. The function has 4 arguments the values of which are available from the array $_POST. Depending on which of the 2 .htm pages has called the script, one or two of the arguments will be empty and coded blank in the hidden input of the form which is used by the object to decide the internal processing.

The call to the object returns an array, $result. The first element, $result[0], can have 3 alternative values, i.e. yes, no or new, determining the further actions. yes results in a message to the visitor that she has been successfully logged in, no generates a message that the username, PIN or possibly both were rejected, and new that the registration form has been processed and username and PIN are available. Instead of the message to the visitor in Line 16, an email can be sent to the web master for manual inspection and approval.

LOGIN class
In the example **Authorization and Authentication**, we demonstrated how we could use functions to make the code more re-usable. The next step is to develop this functionality as an object ready to use for any application requiring a controlled access to the site.

We use the knowledge acquired in the previous section in designing the **class LOGIN** and start by enclosing the class in **class login**(...). The class will need a set of **properties** which we define as **private** variables in **Line 3 - 7**. This implies that these variables are **not available** outside the instance generated by the class and have to be assigned values by a method before they can be used.

The class include **5** methods, **1** public and **4** private functions. The **public function main**() is available from outside (in our example from **application.php**) and is used for assigning values to the private variables, opening a database, and deciding which of the private method to call based on the input values. The method **main**() also returns an array with the wanted results.

The **private** methods are **private function sql_table_exists**, **private function authorize**, **private function authenticate** and **pinAssign** and available only within the instance **myLogin**.

The **public function main**($firstname,$lastname,$email,$pin) has **4** arguments which are obtained from one of the **2** form pages.

**Line 11 -15** opens/creates an **SQLite** database with a table **users**. with the necessary columns. **Line 13** calls a method called **sqlite_table_exists**() which was introduced in session **6**. Note that the table **Users** is slightly different from the table used in session **6** because an object-oriented version is used!

Depending on which page (**index.htm** or **registration.htm**) has been used, **$pin** will be blank or contain a code. This is used in **Line 16** to select the relevant next method to call. If **$pin** is **blank**, it is obviously a new user who wants to be registered and the function **authorize**() is called with the arguments **$db**, **$firstname**, **$lastname** and **$email** in **Line 17**. Note that the reference to the database must also be passed to the function. This function returns an array named **$access**. The content of this array will be discussed in connection with this function.

If on the other hand **$pin** is **not** blank, it is assumed that it is an known visitor and the function **authenticate**() is called with the arguments **$db**, **$email** and **$pin** in **Line 21**. This function returns an array named **$approved** which will be discussed below.

```php
//CLASS LOGIN
1. <php?
2. class login {
3. private $firstname;
4. private $lastname;
5. private $email;
6. private $pin;
```
private $db;

public function main($firstname,$lastname,$email,$pin) {
    $this->pin=$pin;
    $d='users.db';
    $db=new SQLiteDatabase($d);
    if(!$this->sqlite_table_exists($db,'Users')) {
        $db->query("CREATE TABLE Users (firstname VARCHAR(20), lastname VARCHAR(20), email VARCHAR(20), pin VARCHAR(10), visits VARCHAR(3))");
    }
    if ($pin==" ") {
        $access=$this->authorize($db,$firstname,$lastname,$email);
        return $access;
    }
    else {
        $approved=$this->authenticate($db,$email,$pin);
        return $approved;
    }
}

private function sqlite_table_exists($sqlite, $table) {
    $result = $sqlite->query("SELECT name FROM sqlite_master WHERE type='table' AND name='$table'"⋮
    return $result->numRows() > 0;
}

private function authenticate($db,$email, $pin) {
    $email2=sqlite_escape_string($email);
    $pin2=sqlite_escape_string($pin);
    $r= $db->arrayQuery("Select * FROM Users WHERE pin='$pin2'"⋮
    if ($r) {
        foreach ($r as $result) {
            if ($result['email'].$result['pin']==$email2.$pin2) {
                $approved[0]="yes";
                $approved[1]=$result['firstname'];
                $approved[2]=$result['lastname'];
            } else $approved[0]="no";
        }
    } else $approved[0]="no";
    return $approved;
}

private function authorize($db, $firstname, $lastname, $email){
    $email2=sqlite_escape_string($email);
    $r= $db->arrayQuery("Select * FROM Users WHERE email='$email2'"⋮
    if ($firstname==" ") OR $lastname==" ") OR $r) {
        $access[0]="no";
    } else {
        $pin =$this->pinAssignment($db);
        $db->arrayQuery("INSERT INTO Users(firstname, lastname, email, pin) VALUES('$firstname','$lastname','$email', '$pin')"⋮
}
58. $access[0]="new";
59. $access[1]=$email;
60. $access[2]=$pin;
61. }
62. return $access;
63. }
64. //Pin assignment
65. private function pinAssignment($db) {
66. $r= $db->arrayQuery("Select * FROM Users");
67. $used="yes";
68. while ($used=="yes"){
69. srand();
70. $pin2=rand(1000,99999);
71. $used="no";
72. foreach ($r as $row) {
73. if ($row['pin']==$pin2) {
74. $used="yes";
75. break;
76. }
77. }
78. }
79. return $pin2;
80. }
81. }
82. ?>

We start by studying what happens when $pin and authenticate() is called. First, cleaned $email2 and $pin1 versions are obtained by Line 32-33, and a database search for a row with $pin2 initiated. If one or more lines are retrieved, the row elements are stored in an array called $result, and the concatenated $email and $pin compared with the submitted to the test in Line 37. If the test is positive, the $approved[0] is assigned value yes and the following elements with the $firstname and $lastname. Else the $approved[0] is assigned value no.

Assume that $pin is blank and authorize() is called. This is a private function and can only be called from main(). In Line 50 the submitted email is 'cleaned' before the database call in Line 51. A search in the database for a possible email address corresponding to the submitted is done. If found, or if one or both of the name variables are blank, the first array element, $access[0], is assigned the value no.

If the submitted names are not blank and the submitted email unused, a call to the method pinAssign() returns an unused pin. This method we already know as a function discussed previously. Names, email and assigned pin is then inserted into the database in Line 57, and finally the $access array is prepared in Lines 58-60.

The object myLogin returns an array the first element of which is either yes, no or new indicating a positive, negative authentication or new indicating a successful registration. Return to the application.php to see how these and the following elements are used.
Session 10: Course site

Course framework

In this last session of the course, we shall discuss the basics of developing an online course by means of PHP. As a model for an application example, we use a simplification of the present course. Within the limits of a session, the example can only reflect the main parts of the course implementation.

Infrastructure

The general infrastructure uses the course you have been attending as a model, and you will recognize the 3 parts, Information, Sessions and Communication, and their contents as well as a part called Structure. The course system also generates and maintains an SQLite database called users.db.

The course is accessed by an HTML form page coded in index.htm. Student registration is carried out by a small system entered by registration.htm. Organization of authorization and authentication is done by validate.php and the class login. The use of the system is recorded by means of our user-defined function logging(). A small system for viewing the log of the application can be entered by view.php.

As pointed out, the course example is not completely developed, but you will find examples of elements from all the above 4 parts. As usual, the application is available as a live example.

Implementation

The example will make use of the 2 user-developed libraries, myClassLibrary.php and my_functions.php located in the directory /PHPRoot/.

index.htm

The index.htm is an ordinary form page for the student to submit her user name (note that e-mail address is used as user name in this example) and PIN code.

The users can by visiting registration.htm obtain valid user name and PIN code.
7. <body>
8. <center>
9. <h1><font color="Blue">Dynamic web applications with PHP</font></h1>
10. <p>Course access requires that you have been granted admittance by your university/organization. You will then get an accesscode by e-mail from the instructor. </p>
11. <p><b><font color="Red">Access the course with your</font></b></p>
12. <p><b><font color="Red">Access the course with your</font></b></p>
13. <form method="POST" action="validate.php">
14. <table>
15. <tr><td>Your user name:</td> <td><input name="email" type="text"></td></tr>
16. <tr><td>PIN code:</td> <td><input type="Password" name="pin"></td></tr>
17. <input type="hidden" name="firstname" value=" ">
18. <input type="hidden" name="lastname" value=" ">
19. <tr><td>Click button:</td><td><input type="submit" name="response" value="ENTER"></td></tr>
20. </table>
21. </form>
22. </center>
23. </body>
24. </html>

Remember that hidden blank values must be used for firstname and lastname for identification of the relevant HTML page. Line 12 shows that the execution control is transferred to the PHP script validate.php.

**registration.htm**

To attend the course, a student has to be admitted. In the example, an application is sent by means of the form registration.htm and if the provided e-mail address is not already used a PIN is returned immediately. In the real world, an application must be sent the school by a student, and if she is admitted, the school/instructor uses registration.htm to record the student in the course database.
The form is another ordinary HTML form. Note again that a hidden blank value is used for pin. This form also calls on the validation.php for processing.

**validate.php**

The `validate.php` script has several tasks. Line 3 and 4 make the 2 needed libraries available for use by the script. The class library is already used in the next line where an object instance, `$myLogin`, is created. The functionality of this object has already been discussed in a previous session, and we know that it is the object method `main()` which must be called to invoke the object.

Note that all 4 arguments are specified even though we know that `index.htm` only provides 2 data, and registration 3 data. The `$myLogin` object uses the supplied data to determine if it should process a registration, an authentication or an invalid set of data.

```
<tr><td>Your last name:</td> <td><input name="lastname" type="text" SIZE="20" value=""></td>
</tr>
<tr><td>Your e-mail address:</td><td> <INPUT TYPE="text" name ="email" SIZE="20"></td></tr>
<input type="hidden" name="pin" value=" ">
<tr><td>Click the button:</td> <td><INPUT TYPE="SUBMIT" NAME="response" VALUE="Submit"></td></tr>
</table>
</FORM>
<center><foo...
that an invalid set of data was submitted, while "new" informs the calling script that a new student has been registered.

In the case of a valid login, the function logging() is called with the authenticated PIN code as argument. This results in a new record in the application log indicating a successful entrance to the system. The student is then informed that she has been logged in and asked to continue pointing to frame1.htm in the subdirectory structure.

If the data came from the registration page, a user name (student's email address) and an unused PIN code is returned. In the example, it is assumed that the instructor can read these on the screen and forward them to the student. It is possible and more elegant to let the system send the data directly to the student.

**frame1.htm**

We want to present the course in a frame setting with header and footer, a navigation window to the left and a text window to the right.

frame1.htm is a typical HTML set up for such frames. It features 4 frames, a header, a left column for alternative selections, a main frame for texts, illustrations, examples, etc., and a footer.

```
1. <html>
2. <FRAMESET ROWS="9%,*">
3. <FRAME NAME="HEADER" SRC="header.htm" MARGINWIDTH="10" MARGINHEIGHT="10" SCROLLING="no" FRAMEBORDER="1" NORESIZE>
4. <FRAMESET COLS="20%,*">
5. <FRAME NAME="LIST2" SRC="list2.htm" MARGINWIDTH="10" MARGINHEIGHT="10" SCROLLING="yes" FRAMEBORDER="1">
6. <FRAMESET ROWS="*,8%">
7. <FRAME NAME="TEXTS" SRC="../information/welcome/welcome.htm" MARGINWIDTH="10" MARGINHEIGHT="10" SCROLLING="auto" FRAMEBORDER="1">
8. <FRAME NAME="FOOTER" SRC="footer.htm" MARGINWIDTH="10" MARGINHEIGHT="5" SCROLLING="no" FRAMEBORDER="1">
9. </frameset>
10. </FRAMESET>
11. </FRAMESET><noframes></noframes>
12. </html>
```

In the left side frame, navigation options are presented by means of list2.htm. This frame remains unchanged during the visit. The starting content of the right side main frame is implemented in /information/welcome/welcome.htm and is changing according to the selections made during the visit.

**list2.htm**
The navigation within the course is to a large extent made from the options in the left side frame. list2.htm illustrates a possible list of contents and options. The page indicates that only a few items are implemented.

1. <!-- list2.htm -->
2. <H3><FONT COLOR="Blue">Information links:</FONT></H3>
3. <a href="../information/about/about.php" target="TEXTS">About the course</a>
4. <a href="../information/dummy.htm" target="TEXTS">Articles</a>
5. <a href="../information/dummy.htm" target="TEXTS">Assignments</a>
6. <a href="../information/dummy.htm" target="TEXTS">Calendar</a>
7. <a href="../information/dummy.htm" target="TEXTS">Curriculum</a>
8. <a href="../information/dummy.htm" target="TEXTS">FAQ</a>
9. <a href="../information/dummy.htm" target="TEXTS">Grades</a>
10. <a href="../information/dummy.htm" target="TEXTS">Images</a>
11. <a href="../information/dummy.htm" target="TEXTS">Literature</a>
12. <a href="../information/dummy.htm" target="TEXTS">Search engine</a>
13. <a href="../information/dummy.htm" target="TEXTS">Software</a>
14. <a href="../information/dummy.htm" target="TEXTS">Virtual classroom</a>
15. <a href="../information/dummy.htm" target="TEXTS">Web links</a>
16. <H3><FONT COLOR="Blue">Session links:</FONT></H3>
17. <a href="/sessions/session1.php?sessioncounter=1" target="TEXTS">Session 1: Static application with HTML</a>
18. <a href="/sessions/dummy.htm" target="TEXTS">Session 2: Dynamic application with PHP</a>
19. <a href="/sessions/dummy.htm" target="TEXTS">Session 3: Dynamic application without database</a>
20. <a href="/sessions/dummy.htm" target="TEXTS">Session 4: SQLite database</a>
21. <a href="/sessions/dummy.htm" target="TEXTS">Session 5: Application of SQLite</a>
22. <a href="/sessions/dummy.htm" target="TEXTS">Session 6: Functions in PHP</a>
23. <a href="/sessions/dummy.htm" target="TEXTS">Session 7: File processing</a>
24. <a href="/sessions/dummy.htm" target="TEXTS">Session 8: Object-oriented interface in PHP</a>
25. <a href="/sessions/dummy.htm" target="TEXTS">Session 9: Login class</a>
26. <a href="/sessions/dummy.htm" target="TEXTS">Session 10: Course site</a>
27. <H3><FONT COLOR="Blue">Communication links:</FONT></H3>
28. <a href="/communications/static/menu_board.php" target="TEXTS">Message board</a>
29. <a href="/communications/dummy.htm" target="TEXTS">Question and answers</a>
30. <a href="/communications/dummy.htm" target="TEXTS">Progress report</a>
31. <font color="Blue">Subject: </font> <font size="-1">PHP</font>
32. <p></p>
33. <p>a href="/index.htm" target="_parent">Logout</p>

A few examples of items are presented below.

about.php

This script is located in /information/about/about.php. Two questions may be asked. First, why is it a .php script and not an .htm page? The answer is that to take advantage of the logging function, we need the Line 2-5 of PHP statements.
The second question is why the subdirectory /about/ is necessary. The answer is that it is not necessary in our example, but if we wanted to develop any conditions for accessing the \texttt{about.php}, or several alternative versions of the script, it is convenient to store the script containing conditions in the same directory.

```php
1. <!--- about.php --->
2. <?php
3. include("/PHPRoot/my_functions.php");
4. logging($_SESSION['PIN']);
5. ?>
6. <h1><font color="Blue">About Dynamic web applications with PHP 5</font></h1><p></p>
7. <p><b><i><font color="Red">Author: </font></i></b>Svein Nordbotten.</p>
8. <p>The publication of web applications is increasing fast. Still, most web sites are <font color="#FF0000">static</font>, i.e. they are created with a <font color="#FF0000">fixed content</font> and cannot be modified by information provided by the visitor on a returned form or by previously saved information in for example a database.</p>
9. <p><b>PHP</b> is one of several tools which can be used in creating dynamic web applications. Other alternatives are <b>ASP.NET</b>, <b>ColdFusion</b> and <b>iHTML</b> to mention a few. With version <b>PHP 5</b> has introduced several important <font color="#FF0000">new features</font>. It contains a <font color="#FF0000">built in database</font>, <b>SQLite</b>, which makes the installation and connection to a separate database unnecessary. The <b>SQLite</b> is completely integrated in the <b>PHP Language</b> and is easy to use. <b>PHP 5.0</b> also includes <font color="#FF0000">object-orientation</font> as a second interface in addition to the earlier functional interface. This permits creating even more easily <font color="#FF0000">re-useable</font> software than previously.</p>
```

\texttt{session1.php}

Similar to the \texttt{about.php}, the scripts \texttt{session1.php} and \texttt{menu_board.php} are implemented in the example. Only a small piece of \texttt{session1.php} is shown below. As you can see, the script starts with the same \texttt{include()} and \texttt{logging()} functions as does the \texttt{about.php} to record a visitor passing these scripts.

```php
1. <?php
2. include("/PHPRoot/my_functions.php");
3. logging($_SESSION['PIN']);
4. ?>
5. <h2><font color="blue">Session 1: Static web applications</font></h2>
6. <h3><font color="#0000FF">Web applications</font></h3>
7. <p>The topic of this course is the design and implementation of <font color="#FF0000">web applications</font>. In this context a web application is a <font color="#FF0000">server-based system</font> which can interact with the user and respond with several interrelated pages for display at the user’s computer.</p>
```
The end

This is the end of the last session. Thank you for your participation, the author hopes you have enjoyed the course!
A bibliography for further studies

IMPLEMENTING DYNAMIC WEB-SITES:


Hatfield, B. (1999): Active Server Pages for Dummies. IDG. CA.


