8.Introduction to Web Applications

This chapter introduces the concepts necessary to create web-based database applications. On the web, documents are usually created using HTML; HTML pages are static documents, so we need a programming language that can generate HTML pages; here we use PHP as that programming language; we also cover how to use PHP to send SQL commands to a DBMS, specifically to postgreSQL.

In section 8.1, we give a brief introduction to HTML; section 9 (sorry about the numbering :) is a brief introduction to PHP, including how to use PHP to access databases.

8.1. HTML

HTML stands for HyperText Markup Language; it is the language used to create web pages. It is NOT a programming language, but rather a document markup language; this means we write mostly text, with some special code to mark up the document's structure or certain special features of the text (bold, italics etc).

There are several version of HTML, but here we will cover only the most basic features, basically corresponding to HTML 3.2. This is NOT a comprehensive guide to HTML, just a quick start.

In HTML, we use tags to mark up the document. HTML tags are surrounded by < >, for example the tag for bold is <b>. Many tags mark up only a certain portion of text, and we need to mark when they stop being in effect; we call this *closing* the tag, and we usually use the same tag, but ending in /> rather than >; for example, to mark a small section of text as bold we would do: <b>bold text</b>.

Spacing (and line breaks etc) are NOT significant in HTML, so we need to mark where do paragraphs start and end etc. Originally, HTML provided the <br> tag to signal a line break, and the <p> tag to signal the beginning of a new paragraph. Nowadays, they are considered bad style, and using <p> and </p> around each paragraph is considered better form.

Notice that HTML struggles between specifying the ideas (this word should be emphasized, this phrase is a title) and specifying the actual format (this word should be bold, this phrase should be centered and in 14 points). Each browser can render many of the tags in a different way. Cascading Style Sheets (which we don't get into here) can alleviate this problem.

8.1.1. Basic Document Structure

The basic structure of an HTML document is as follows:

```
<html>
<head>
    <title>  put the title here </title>
</head>
<body>
here you write the stuff
</body>
</html>
```

The whole document is surrounded by an <html> tag (closed with </html>), the document is composed of two sections; the head and the body. The title of the document, along with other metadata goes inside the head (the title is usually displayed on the title bar of your browser), and the body contains the actual data of the document.

8.1.2. Basic formatting tags

- `<h1>` is used to denote a level 1 header (and we have h2, h3 ...).
- `<b>` bold
- `<it>` italics
- `<p>` new paragraph. Can close or not
- `<br>` line break

8.1.3. Images

We can also insert images within the text; we do this by using the <img> tag; we use the src parameter within the tag to specify the URL of the image we want to include. For example, if we have an image called logo.jpg in the same folder on the server as our html page, we could insert it into our document by typing: `<img src="logo.jpg">`

8.1.4. Hyperlinks

We often want to create references to other documents (hyperlinks) within an HTML document; we do this by using the <a> tag. Inside the a tag, we need to specify where does the link point to; we do that by adding an href attribute; everything that goes between the <a> and the </a> will be displayed differently (usually in blue and underlined) by the browser, and clicking on that portion of the document will cause the browser to load the other document. So, to have a hyperlink that points to http://spsu.edu and have the words 'Georgia's Technology University' inside, we would write:

```
<a href="http://spsu.edu">Georgia's Technology University</a>
```

Notice that we can nest the tags into each other, so if we wanted to have the logo above, and have it point to spsu.edu we would type:

```
<a href="http://spsu.edu">
    <img src="logo.jpg">
</a>
```

Another attribute you may use in an `<a>` tag is the target attribute, which is important when using frames (described later).

8.1.5. Lists

I tend to do lists a lot. In HTML you use a tag for defining the list, and another tag for each of the items. For the list, you use either <ol> for an ordered list, or <ul> for an unordered list; each list item should be wrapped inside an <li> tag.

Sample HTML	Visual result (possible)

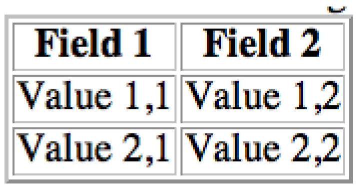
`<ul>`
`     <li>Unordered (ul tag)</li>`
`     <li>Ordered (ol tag)</li>`
`</ul>`
	● Unordered (ul tag)
	● Ordered (ol tag)

8.1.6. Tables

A table is started with the <table> tag. Then, you start each row with the <tr> tag (for table row) and each cell with <td> (for table data).

For example, the html for a table with three rows (the header row and two data rows) would look like this:

Sample HTML	Visual result (possible)

`<table border="2">`
`<tr>`
`    <th>Field 1</th>`
`    <th>Field 2</th>`
`</tr>`
 	
`<tr>`
 	
`    <td>Value 1,1</td>`
`    <td>Value 1,2</td>`
`</tr>`
 	
`<tr>`
`    <td>Value 2,1</td>`
`    <td>Value 2,2</td>`
`</tr>`
	
	Field 1 **Field 2**
	Value 1,1 Value 1,2
	Value 2,1 Value 2,2
	Figure 1: Table as displayed by Firefox on Mac

8.1.7. Forms

HTML Forms allow you to create a web page in which the final user can add information in certain input fields, and can send that information to a program (for example a PHP page), that would run on a server.

An HTML form needs to be enclosed in a `<form>` tag. This tag has a couple of parameters you can use ; method and action. The action parameter should contain a URL specifying which program gets executed on the server. The method specifies how to pass the information, and should be either GET or POST.

Inside the form tags, you can add normal HTML, plus input fields. Each input field is created by an `<input>` tag, with the kind of input field determined by its type argument (older versions of html used a different tag per input field, those tags may still be used). You need to attach a name parameter to each input field, so it can be distinguished from the others.

8.1.7.1. Input fields

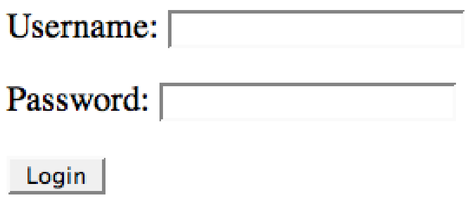
- The simplest input field is of type **text**. You write something like: `<input type="text" name="var1">` , and a field will be inserted. Notice that the input tag creates ONLY the field, and does NOT display the name of the field or anything like that.
- An input field of type **password** works basically the same as text, but all characters typed appear as * rather than the character (the field contains the info you type, it is just *displayed* as *s)
- An input field of type **submit** generates a button that, when clicked, *submits* the form, that is, makes the browser send the data to the server program and displays the results send by the program.
- Other kinds of forms include **radio** for radio buttons, **checkbox** for checkboxes, and **hidden** for

passing information to the server program but NOT displaying it in the browser.

8.1.7.2. A simple example

The HTML code below, will generate a form like the one on the right. Notice we have three input fields, and also notice that the caption displayed before the field is NOT generated automatically, but we need to add it to the HTML code, and is NOT necessarily tied to the name of the field, although it is good practice; in this example, the caption is NOT exactly the same as the name of the field, since the field names are in lowercase, and the caption includes an extra : .

Sample HTML	Visual result (possible)

`<form method="GET" action="something.php">`
`<p>`
`  Username:`
`  <input type="text" name="username">`
`</p>`
`<p>`
`  Password:`
`   <input type="password" name="password">`
`</p>`
`<input type="submit" value="Login">`
`</form>`
	
	Figure 2: Form as displayed by Firefox on Mac.

8.1.7.3. GET vs POST

For sending the form, you can choose to use either the GET method or the POST method. With the GET method, the data in the form gets appended to the URL in the action field, so the data submitted is visible. This represents a security risk, but on the other hand allows you to bookmark the page with the submitted data, or even write an external link to it. By copying the syntax, you can encode data to be sent to a server page in a link, with the data coming from a database, or some other program.

Besides the URL, data sent through a GET request has some size limitation, so when sending big amounts of data you need to use a POST request.

8.1.8. Frames

HTML supports the definition of *frames*; one HTML page can define more than one frame, and each frame may be displaying a separate page; this enables us to change only parts of the page being displayed, and to keep other parts (like menus) always visible; frames have some accessibility issues and make it hard to use the browser's back button, but they provide for an easy way to give the user global menus.

You define frames with the `<frameset>` tag; as an attribute of this tag, you specify either rows (if the frames are shown vertically) or cols (if the frames are shown horizontally); as the value of rows (or cols) you write a comma-separated list of sizes, which may be expressed in pixels or in terms of percentages. Inside the <frameset> tag, you specify frames. For each frame, you specify the content to be displayed in its src attribute (you can specify other attributes, like borders etc); it is also a good idea to specify its name, so you can refer to it.

A common example would be to have a menu on the left, with the contents on the right; you can do it with a frame document like this:

Sample HTML	Visual result (possible)
`<frameset cols="100,*">` `<frame name="menu" src="menu.html">` `<frame name="main" src="main.html">` `</frameset>`	

Notice how we specify cols in the frameset, meaning that the frames will be laid out horizontally (so each frame is a column); we specify that the first frame will have a length of 100 pixels, and the next frame will use the rest of the space (represented with the *).

8.1.8.1. *Links and Frames*

Hyperlinks within a frame will, by default, make *that frame* display the contents of the link; we can modify the link so that it makes other frames changes, or the whole page. We do this by specifying which frame to change in its *frame* attribute; if we specify another frame's name, then that frame will be changed, and if we specify _top then the target will be the main page.

8.1.9. CSS

CSS stands for Cascading Style Sheets; it is a way of separating the formatting of an HTML document from its appearance; learning the basics of CSS will enable you to make your applications visually appealing and consistent with little effort.

You can embed CSS content in your HTML file, but having it as a separate file will enable you to reuse your CSS and to change the appearance of all your pages by changing just one file. To activate a css file for a particular HTML file you add a link tag to its head portion; the link tag has an attribute **`rel`**, which should be set to stylesheet, and another attribute, **`href`** which should have the URL of your css file.

The following sample shows how to point to a css file called mystyle.css (I'm including the whole **`head`** section)

Sample HTML
`<head>` `<title>HTML Examples</title>` `<link rel="stylesheet" href="mystyle.css">` `</head>`

Within a css file, you specify rules that describe how specific things should be displayed. Each CSS rule has a selector and a declaration, the selector specifies which kinds of HTML objects are affected and the declaration specifies the properties that would be applied. For example the following rule:

```
body {background-color:#b0ffb0; }
```

specifies that the background color of the document would be a light green. The selector of the rule is **`body`**, and it specifies that this applies to the body tag; the declaration of the rule, between **`{ }`**, specifies which properties get applied, in this case, **`background-color`** , notice that we use **`:`** instead of =, and that each property 'assignment' ends with a semicolon **`;`**.
Notice also how we specify colors; they start with #, then follow 3 hexadecimal numbers (two hex-digits each), one representing the red intensity, the next one the green and the next one the blue (so

basically, RGB, expressed in hex, values go from 0 – 255 for each component).

another example would be:

```
h1 {text-align: center;}
```

which would specify that for the `<h1>` tag, the alignment would be centered.

We could use other kinds of selectors, and many different properties; for example, I usually do this CSS for my tables:

```
table {border: 2px solid black;}
.odd  {background-color: #b0d0ff;}
.even  {background-color: #ffffff;}
```

The first rule specifies that for tables we want a solid black border, 2 pixels wide. The next two declarations go in parallel; the first one specifies that any object with class odd should have a light blue background and any object of class even should have a white background; I then use the class attribute in my `<td>` tags, alternating (so the first one is `<td class="even">` and the next one `<td class="odd">` etc.

8.2. Introduction to PHP

PHP is a language designed for web development (but can function as a general-purpose scripting language too).

PHP is a dynamically typed language. This means we do NOT need to declare variables, and variables can have values of different types at different times (a variable does NOT have a type per se, but its value does).

In its basic usage for web development, we just embed php code inside an html file (we usually need to give the file a .php extension rather than a .html one), and the web server executes that php code and embeds its **output** in the html file that it sends to the browser.

8.2.1. Embeding into HTML

To embed php code into HTML, we use `<?php` to start the code and `?>` to stop php code and go back to embedding (notice that usually the file needs to have the extension .php).

8.2.2. Variables

PHP code looks much like C (and since many other programming language, like Java and C# also look like C, it looks like those languages), except that any variable reference is denoted with a dollar sign ($).

So, if in C we would write:

```
a=3; or  a=b+c;
```

in PHP we write

`$a=3;` or `$a=$b+$c;`

Notice that, since PHP is a dynamically typed language, we do NOT need to declare the variables (so you don't need to do `int a;` or such); moreover, variables do not have types, only values do, which means you can do `$a=3`, and so `$a` contains a number, and later `$a="Joe"`, so `$a` contains a string (try doing that in java ! :)

8.2.3. Printing

The syntax for calling a function is the same as in C, we write the name of the function and then the arguments inside parenthesis. Although strictly they are not functions, we can use `print` and `echo` as functions to print something. So if we wanted to print the value of the variable name, we would write:

`print($name);` or `echo($name);`

8.2.4. Strings and interpolation

Unlike C, but like most scripting languages, PHP allows for **string interpolation**; that is, in a double-quoted string, variable references are substituted by the value of the variable. For example, after the following php code:

```
$name=”Orlando”;
$greeting=”Hello $name”;
```

The value of $greeting is “Hello Orlando”. Notice that if we use single-quotes to denote a string constant, no interpolation is done (so if we do `$greeting='Hello $name';` then $greeting will contain the string `”Hello $name”`.

We can also concatenate two strings using the period operator (.), so if we do `$a=”Orlando ”. ”Karam”` then $a will contain `”Orlando Karam”.`

Since we use PHP to generate HTML code, and HTML uses the double-quote character (”) as part of its syntax, a common problem is to generate strings that contain the double-quote character themselves; we normally do this by escaping it (adding a \ before the ”)within the string, so if we wanted to have a string that contained Joe surrounded by double-quotes, we could write `”\”Joe\””`. BTW, another problem is that word-processors tend to want to use the typographically-correct quotes, so the character opening the quote looks different (and is actually a different character) than the one closing it, which would cause a syntax error; the common solution is not to use a word-processor for editing code :), I've tried to be careful within this document, but my word-processor may have inadvertently changed some of my characters.

8.2.5. Arrays

PHP provides for arrays, with the usual notation for indexing; however, in php arrays are closer to Vectors in Java or C++, since they can grow. An easy way to create an empty array is by calling the `array()` function. Notice that the array initialization syntax is not the same as C or Java, we can't do `$a={1,2,3}`, instead we need to do `$a={0 =>1, 1=>2, 2=>3}`. Notice that PHP won't complain if we use a non-existent index for the array, but it will just create an entry for that index.

8.2.6. Associative arrays

Like most scripting languages, PHP provides associative arrays; that is arrays whose index is not an integer, but a string ; they are roughly equivalent to Maps in C++ or Java.

So we can do:

```
$arr[”hello”]=”world”
```

and later

```
print ($arr[”hello”]);
```

To check whether an element for that key exists in an array, you can use the `isset` function.

8.2.7. Control Structures

PHP supports the usual control structures, with standard C syntax (although they may look a little weird with the extra $ s). As in C, a block can be one statement, or several statements enclosed in braces {}.

8.2.8. If/else

We can express conditionals with the if/else construct. We write **if**, then a condition in parenthesis, and a block to be executed if the condition is true. Additionally, we may specify a block to be executed if the condition is false, with the **else** keyword.

Here is an example of an if statement:

```
if ($age>=18) {
     print(”You are old enough to vote”);
} else {
     print(”You are NOT old enough to vote yet”);
}
```

8.2.9. while loop

PHP supports a while loop, with syntax similar to C (while, condition in parenthesis and then a block).

8.2.10. for loop

PHP also supports a for loop, like C. The for loop has 3 expressions, separated by semicolons; the first expression initializes one or more variables, the next one is a comparison to be done before each pass, and the final one is an increment. The syntax looks slightly weird due to the $. For example, to print the numbers from 0 through 10, we would do:

```
for ($i=0; $i<10;++$i)
     print $i;
```

8.2.11. Functions

You can define your own function in php; you do that by using the keyword function, then the name of the function and then a list of parameters in parenthesis. For example, to define a function to add two numbers you'd do:

```
function add($a, $b)
{
     return $a+$b;
}
```

8.2.12. Including files

Many times we want to factor commonly used code into its own file, and then just include that file when needed (especially since PHP is interpreted and so needs to have all the code available; in a way, we need something to replace import in Java). We can use the functions include or include_once, with the name of the file, i.e. to include a file called functions.php we would do `include_once('functions.php')`.

8.2.13. Objects and Classes

Although they're not required, PHP also supports creating classes and objects. A discussion of PHP's object capabilities is outside the scope of this tutorial; however, one feature we will use is the ability to access fields within an object; for this, PHP uses ->, rather than the . used in C++ or Java; so if `$a` is an object with a field called age, we can increment the age by doing: `$a->age=$a->age+1`.

8.2.14. Forms

PHP supports easy access to form data. It defines three arrays, _GET for data coming through a GET request, _PUT for data coming through a PUT requests, and _REQUEST for data coming either way. We use the name of the variable in the form as the key to the array; so to get the value in the field called age of the form into our own variable called myage, we would write:

`$myage=$_REQUEST[”age”];` or, since we don't really need to interpolate anything in the string, we could also write `$myage=$_REQUEST['age'];`

8.3. Database access in PHP

PHP provides a nice set of functions to access many databases, and specifically postgresql. PHP provides an abstract layer for accessing many DBMSs through the same interface, called PEAR DB; but for simplicity, we will cover only the functions that are postgresql specific (if you were to switch to another DBMS you'd probably need to change the first few characters of each function's name).

The functions we will use are:

Function	Purpose
pg_connect	establishes a connection to the database and returns a handle to it
pg_query, pg_query_params	executes a query and returns a handle to the result set; notice the query can be an INSERT, UPDATE or DELETE, besides a SELECT. pg_query_params is used for parametrized queries, that is, those for which the final form is obtained by interpolating strings or otherwise incorporating variables into a query string.
pg_numrows	returns the number of rows in a result set
pg_fetch object	returns an object representing a row in a result set

8.3.1. pg_connect

The pg_connect function takes a string containing connection parameters, and returns a handle to the connection. The string that you pass as arguments contains name=value pairs, separated by spaces. The arguments you give are:

- host -- name of the host normally = localhost, meaning the same server as the website
- dbname – name of the database. Normally, same as your username for your own db
- user – name of the user, normally, your spsu email handle for our server
- password – your password

Example:

```
$conn=pg_connect(“host=localhost dbname=ok user=ok password=abc123”);
```

8.3.2. pg_query

This function takes a connection handle and a string representing a SQL query, and returns a handle to the result set. Notice this function also works for insert/update etc although it could impose a security risk.

Example:

```
$res=pg_query($conn,”SELECT * FROM Student”);
```

Actually, the connection handle is optional; if not provided, it uses the last connection obtained through pg_connect, so we can also write:

```
$res=pg_query(”SELECT * FROM Student”);
```

Notice the value returned by pg_query is a *handle* to the result set; that is an opaque value (think an object, without the nice syntax) that somehow contains all the rows returned. We would call functions like `pg_num_rows` and `pg_fetch_object` to extract information from that result set.

8.3.3. pg_query_params

This function takes a string representing a SQL query with special markers for parameters, and an array containing values for those parametesrs, and returns a handle to the result set. Notice this function also works for insert/update etc.

Example:

```
$age1=20; $age2=20;
$res=pg_query_params(
          ”SELECT * FROM Student WHERE age>$1 AND age <$2”,
          array($age1,$age2)
     );
```

Again, the value returned is a handle to the result set, same as from pg_query.

Notice you *could* use pg_query, and pass it a string that is generated from other variables, but this can lead to serious security issues, which is why we recommend using pg_query_params instead.

Also, notice that pg_query and pg_query_params can both be used to execute any kind of SQL queries, including UPDATE, INSERT and DELETE.

8.3.4. pg_num_rows / pg_numrows

This function takes a handle to a result set and returns the number of rows in it. For now, both functions exist (with or without an underscore between num and rows). With the underscore is newer syntax, and more consistent with the naming convention used for the other functions. In the future, pg_numrows will probably disappear.

8.3.5. pg_fetch_object

This function takes a handle to a result set and a number, and returns a PHP object with the same data as the corresponding row in the result set. The first row is row number 0. After you get the object, you can use the -> operator to access each one of its fields. All field names will be in lowercase !!!. There are similar functions that return the row as an array or associative array.

Example:

```
$obj=pg_fetch_object($res,0);

print $obj->name;
```

8.3.6. Example for retrieving all rows:

```
$conn = pg_connect("host=localhost user=ok dbname=ok password=abc");
```

This line establishes the connection (notice normally the database names is the same as the user name, and so both parameters get assigned the same value on the string), and stores (a handle to) that connection in the $conn variable.

```
$query_str="Select * FROM student";
```

This line just initializes a string variable called $query_str. Notice that the value of that string variable is a SQL statement.

```
$res=pg_query($conn, $query_str);
```

This line actually sends the query to the DBMS (postgresql in this case). It uses the connection already established ($conn) and the query stored in $query_str, and stores (a handle to) the rows returned by the DBMS as a result of the query in $str (I tend to call tis set of rows the result set). We could have also used directly a string constant instead of $query_str, but if we make a mistake in our query string and get an error message from the DBMS, we can simply insert a print statement, which helps with debugging. Also, if we do not include a connection, pg_query uses the last connection established with pg_connect.

Now, we're going to use a for loop to iterate through all the rows returned, and print their values. We will use pg_num_rows to find out how many rows are there, and pg_fetch_object to fetch each individual row.

```
for($i=0; $i<pg_num_rows($res); ++$i) {
```

Start $i at 0, go until the last row in the result set ($res), increment by one.

```
    $row=pg_fetch_object($res,$i);
```

Fetch the i-th row from $res, as an object, store it in the variable $row.

```
    print("$row->name<br>\n");
```

Print the field called name, from that row.

}

8.3.7. Other comments

- http://php.net has much information about php; in particular, http://www.php.net/manual/en/ref.pgsql.php has a reference for all the functions to connect with postgresql.

- Note that php is case sensitive (although SQL is not), and also that the postgresql driver converts all field names to lower-case in the results of a query; this means all the fields in objects obtained through `pg_fetch_object` must be in lower case.

9. Web applications in PHP

Your application will be conceptually organized into pages; it is a good idea to keep each page in its own file. To implement a piece of functionality, you will usually need two pieces:

1. An HTML form
2. A PHP page called from that form, that uses the input provided by the form to execute one or more SQL statements.

Now, how do we relate all those separate pages ? By using links; many of the pages can include links to each other (and this links may even be generated using PHP, and so the number of links may be dependent on the information on the database; recall that we can actually encode information in a link by adding ? at the end, and then name=value pairs); if we define a menu in a frame, that menu can link to several pages so the user can keep track of the functionality, while the functionality being currently accessed is displayed in another frame.

Notice that each PHP page will be a separate page, and that each request comes as a completely separate request; however, many times, we want to give the user the illusion that they are accessing an application, and that the pages know what other pages that particular user has accessed recently; within web apps, we call this idea of all the recent interactions of a user with a web site, a *session*; PHP supports keeping track of user sessions, which is explained in the section below.

10. Sessions in PHP

When doing web applications, each page request is sent individually, and requests for different browsers can come interleaved; however, we usually want to give the individual users the illusion that the server knows them, and that they're running one application, instead of sending a series of page requests.

HTTP, the network protocol normally used to transfer web pages, allows for the use of *cookies*, that is, name-value pairs that a server can ask a browser to keep; the browser then sends those cookies to the server whenever it requests a page from it (or from the same folder of that server).

Although cookies are tremendously useful, a web application cannot store much information using them due mainly to performance and security issues; if we store several cookies on a browser, they are transferred on each page request, with the added network load; also, since they're stored on the browser, a malicious user could change the stored cookies and fool the server into believing false information.

The standard solution is to use session variables; the application server generates a random-looking session id, and sends a cookie to the browser with that session id; all variables associated with that session are stored by the application server, not by the client (in our case, we simplistically think of 'php' as the application server, in reality it is usually a module (mod_php) running within the web server (apache) that keeps that acts as the application server).

A further complication is that the HTTP protocol specifies that cookies from the server need to be sent *before* any actual html content; so the PHP functions for dealing with cookies need to be sent before any output.

10.1.1. Session variables

Using session variables in PHP is very easy; for each page, we need to call the function session_start(), being careful that this call occurs before any output (any html, any echo or print, even whitespaces before the <? tag); after doing this, we have a $_SESSION associative array, into which we can write variables ($_SESSION['abc']='def') or read from it (print $_SESSION['abc']). This session variables (the $_SESSION array) are passed from page to page, and the app server makes sure the right session variables get associated with each session.

10.1.2. A simple session example

The following page just sets the session variable test to the value abc123; it also displays a link to a second page, disp_session.php

```
<?php
// always call, before ANY output
session_start();
// this sets the session variable test, to the value abc123
$_SESSION['test']="abc123";
?>
<html>
<head>
  <title> Simple session (first)</title>
</head>
<body>
Session variable test created.
<a href="disp_session.php">Test it</a></body>
</html>
```

And the page that reads the session variable, disp_session.php would be as follows:

```
<?php
// always call, before ANY output
session_start();
// this reads the session variable test
$test=$_SESSION['test'];
?>
<html>
<head>
  <title> Simple session (second)</title>
</head>
<body>
The session variable test is: <?= $test ?>
```

10.1.3. Patterns of use for Sessions

The most common pattern would be to have a login form, which sends the username and password to a php page, say login.php; this page verifies that the username/password combination is valid, and if so, it sets a session variable with the username or user id if different (if invalid, displays some error message). Other pages in the application will check whether the session variable is set, and if not set (so the user hasn't logged on) will redirect the user to the login page, or will display different information depending on the status.

For example, in an auction application, the pages could display info about categories and items, but include a 'Buy This' button only if the user is logged on.