**LAB EXPERIMENT FILE**

**1.CREATING AN HTML WEB PAGE FORMS**

<!DOCTYPE html>

<html>

<head>

<title>WEB PAGE</title>

</head>

<body>

<form>

<label>Name::</label><input type="text" name="name"><br>

<label>Email::</label><input type="email" name="email"><br>

<label>Password::</label><input type="Password" name="Password">

<label>Address::</label><input type="text" name="name"><br>

<label>Gender::</label>Male<input type="radio" name="x">

<label></label>Female<input type="radio" name="x">

</form>

</body>

</html>

**2.CREATING HOME PAGE USING HTML**

<!DOCTYPE html>

<html>

<head>

<title>Homepage</title>

<style type="text/css">

img{

width: 100%;

height: 100%;

}

li{

float: left;

padding-top: 10px;

padding-bottom: 6px;

padding-left: 25px;

}

</style>

</head>

<body>

<img src="logonew.png">

<div class="nav">

<nav>

<ul>

<li><a href="#">HOME</a></li>

<li><a href="#">ABOUT</a></li>

<li><a href="#">SERVICES</a></li>

<li><a href="#">PLACEMENTS</a></li>

<li><a href="#">CONTACT</a></li>

<li><a href="#">SEARCH</a></li>

</ul>

</nav>

</div>

</body>

</html>

**3.CREATING XHTML AND CSS AND UNDERSTANDING ITS USE IN CREATING WEB PAGES**

<!DOCTYPE html>

<head>

<meta charset="UTF-8">

<meta name = "rating" content = "general" />

<meta name = "robots" content = "noindex, nofollow" />

<title>PUT YOUR TITLE HERE</title>

<style type = "text/css">

Place your cascading style information here

</style>

</head>

<body>

<p>PUT YOUR TITLE HERE</p>

<p>Write a paragraph about yourself here.</p>

<p>This is your list</p>

<ol>

<li>item 1(remove all of information , item1, item2, etc., and replace with your own)</li>

<li>item 2</li>

<li>item 3</li>

<li>item 4</li>

<li>item 5</li>

</ol>

Put in your picture. <img src = "myPicture.jpg" />

Two hyperlinks

<a href = "http://www.my favorite website.com">My Favorite Website</a>

<a href = "http://www.my favorite website.com">My Favorite Website</a>

This is the table (it is a 3 X 4 grid)

<table>

<tr>

<td>1 </td>

<td>2 </td>

<td>3 </td>

</tr>

<tr>

<td>1 </td>

<td>2 </td>

<td>3 </td>

</tr>

<tr>

<td>1 </td>

<td>2 </td>

<td>3 </td>

</tr>

<tr>

<td>1 </td>

<td>2 </td>

<td>3 </td>

</tr>

</table>

</body>

</html>

**4.SETTING UP AND CONFIGURATION OF APACHE TOMCAT SERVER**

**TOMCAT SERVER CONFIGURATION**

There are certain steps we must follow for configuring Apache Tomcat 7.

**Step 1**

Download and Install Tomcat 1. Go to http://tomcat.apache.org/download-70.cgi then go to the Binary Distribution/Core/ and download the "zip" package (for example "apache-tomcat-7.0.40.zip", about 8MB).

**Step 2**

Check the installed directory to ensure it contains the following sub-directories:

bin folder

logs folder

webapps folder

work folder

temp folder

conf folder

lib folder

**Step 3**

Now, we need to create an Environment Variable JAVA\_HOME. We need to create an environment variable called "JAVA\_HOME" and set it to our JDK installed directory. 1. To create the JAVA\_HOME environment variable in Windows XP/Vista/7 we need to push the "Start" button then select "Control Panel" / "System" / "Advanced system settings". Then switch to the "Advanced" tab and select "Environment Variables" / "System Variables" then select "New" (or "Edit" for modification). In "Variable Name", enter "JAVA\_HOME". In "Variable Value", enter your JDK installed directory (e.g., "c:\Program Files\Java\jdk1.7.0\_{xx}"). 2. For ensuring that it is set correctly, we need to start a command shell (to refresh the environment) and issue: set JAVA\_HOME JAVA\_HOME=c:\Program Files\Java\jdk1.7.0\_{xx} <== Check that this is OUR JDK installed directory 3. Sometimes we need to set JRE\_HOME also. So for creating JRE\_HOME we need to use the same procedure. Push the "Start" buttonthen select "Control Panel" / "System" / "Advanced system settings". Then switch to the "Advanced" tab and select "Environment Variables" / "System Variables" then select "New" (or "Edit" for modification). In "Variable Name", enter "JRE\_HOME". In "Variable Value", enter your JRE installed directory (e.g., "C:\Program Files\Java\jre7\").

**Step 4**

Configure Tomcat Server

The configuration files of the Apache Tomcat Server are located in the "conf" sub-directory of our Tomcat installed directory, for example "E:\myserver\tomcat7.0.40\conf". There are 4 configuration XML files:

1. context.xml file 2. tomcat-users.xml file

**Step 5**

Now, start the tomcat server Executable programs and scripts are kept in the "bin" sub-directory of the Tomcat installed directory, e.g., "E:\myserver\tomcat7.0.40\bin".

**5.UNDERSTANDING MODIFICATION OF WEB.XML**

**Understanding modification of Web.XML**

The Deployment Descriptor: web.xml

Java web applications use a deployment descriptor file to determine how URLs map to servlets, which URLs require authentication, and other information. This file is named web.xml, and resides in the app's WAR under the WEB-INF/ directory. web.xml is part of the servlet standard for web applications.

For more information about the web.xml standard, see the [Metawerx web.xml reference wiki](http://wiki.metawerx.net/wiki/Web.xml) and [the Servlet specification](https://jcp.org/aboutJava/communityprocess/final/jsr315/index.html).

Deployment descriptors

A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve web requests. When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.

The deployment descriptor is a file named web.xml. It resides in the app's WAR under the WEB-INF/ directory. The file is an XML file whose root element is <web-app>.

Here is a simple web.xml example that maps all URL paths (/\*) to the servlet class mysite.server.ComingSoonServlet:

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee  
         http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  
         version="3.1">  
    <servlet>  
        <servlet-name>comingsoon</servlet-name>  
        <servlet-class>mysite.server.ComingSoonServlet</servlet-class>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>comingsoon</servlet-name>  
        <url-pattern>/\*</url-pattern>  
    </servlet-mapping>  
</web-app>

**Note:** If you factor your application into services, each service has its own configuration parameters.

Servlets and URL paths

web.xml defines mappings between URL paths and the servlets that handle requests with those paths. The web server uses this configuration to identify the servlet to handle a given request and call the class method that corresponds to the request method. For example: the doGet() method for HTTP GET requests.

To map a URL to a servlet, you declare the servlet with the <servlet> element, then define a mapping from a URL path to a servlet declaration with the <servlet-mapping> element.

The <servlet> element declares the servlet, including a name used to refer to the servlet by other elements in the file, the class to use for the servlet, and initialization parameters. You can declare multiple servlets using the same class with different initialization parameters. The name for each servlet must be unique across the deployment descriptor.

    <servlet>  
        <servlet-name>redteam</servlet-name>  
        <servlet-class>mysite.server.TeamServlet</servlet-class>  
        <init-param>  
            <param-name>teamColor</param-name>  
            <param-value>red</param-value>  
        </init-param>  
        <init-param>  
            <param-name>bgColor</param-name>  
            <param-value>#CC0000</param-value>  
        </init-param>  
    </servlet>  
  
    <servlet>  
        <servlet-name>blueteam</servlet-name>  
        <servlet-class>mysite.server.TeamServlet</servlet-class>  
        <init-param>  
            <param-name>teamColor</param-name>  
            <param-value>blue</param-value>  
        </init-param>  
        <init-param>  
            <param-name>bgColor</param-name>  
            <param-value>#0000CC</param-value>  
        </init-param>  
    </servlet>

The <servlet-mapping> element specifies a URL pattern and the name of a declared servlet to use for requests whose URL matches the pattern. The URL pattern can use an asterisk (\*) at the beginning or end of the pattern to indicate zero or more of any character. The standard does not support wildcards in the middle of a string, and does not allow multiple wildcards in one pattern. The pattern matches the full path of the URL, starting with and including the forward slash (/) following the domain name. The URL path cannot start with a period (.).

    <servlet-mapping>  
        <servlet-name>redteam</servlet-name>  
        <url-pattern>/red/\*</url-pattern>  
    </servlet-mapping>  
  
    <servlet-mapping>  
        <servlet-name>blueteam</servlet-name>  
        <url-pattern>/blue/\*</url-pattern>  
    </servlet-mapping>

With this example, a request for the URL http://www.example.com/blue/teamProfile is handled by the TeamServlet class, with the teamColor parameter equal to blue and the bgColor parameter equal to #0000CC. The servlet can get the portion of the URL path matched by the wildcard using the ServletRequest object's getPathInfo() method.

**Note:** Static files that are served verbatim to users, such as images, CSS or JavaScript, are handled separately from paths mentioned in the deployment descriptor. A request for a URL path that matches a path to a file in the WAR that's considered a static file will serve the file, regardless of servlet and filter mappings in the deployment descriptor. You can exclude files from those treated as static files using the [appengine-web.xml](https://cloud.google.com/appengine/docs/standard/java/config/appref) file.

The servlet can access its initialization parameters by getting its servlet configuration using its own getServletConfig() method, then calling the getInitParameter() method on the configuration object using the name of the parameter as an argument.

String teamColor = getServletConfig().getInitParameter("teamColor");

JSPs

An app can use JavaServer Pages (JSPs) to implement web pages. JSPs are servlets defined using static content, such as HTML, mixed with Java code.

App Engine supports automatic compilation and URL mapping for JSPs. A JSP file in the application's WAR (outside of WEB-INF/) whose filename ends in .jsp is compiled into a servlet class automatically, and mapped to the URL path equivalent to the path to the JSP file from the WAR root. For example, if an app has a JSP file named start.jsp in a subdirectory named register/ in its WAR, App Engine compiles it and maps it to the URL path /register/start.jsp.

If you want more control over how the JSP is mapped to a URL, you can specify the mapping explicitly by declaring it with a <servlet> element in the deployment descriptor. Instead of a <servlet-class> element, you specify a <jsp-file> element with the path to the JSP file from the WAR root. The <servlet> element for the JSP can contain initialization parameters.

    <servlet>  
        <servlet-name>register</servlet-name>  
        <jsp-file>/register/start.jsp</jsp-file>  
    </servlet>  
  
    <servlet-mapping>  
        <servlet-name>register</servlet-name>  
        <url-pattern>/register/\*</url-pattern>  
    </servlet-mapping>

**Note:** The **<jsp-file>** must start with a forward slash (**/**) if the JSP is in the application's root directory.

You can install JSP tag libraries with the <taglib> element. A tag library has a path to the JSP Tag Library Descriptor (TLD) file (<taglib-location>) and a URI that JSPs use to select the library for loading (<taglib-uri>). Note that App Engine provides the [JavaServer Pages Standard Tag Library](http://java.sun.com/products/jsp/jstl/) (JSTL), and you do not need to install it.

    <taglib>  
        <taglib-uri>/escape</taglib-uri>  
        <taglib-location>/WEB-INF/escape-tags.tld</taglib-location>  
    </taglib>

Security and authentication

An App Engine application can use Google Accounts for user authentication. The app can use [the Google Accounts API](https://cloud.google.com/appengine/docs/standard/java/users) to detect whether the user is signed in, get the currently signed-in user's email address, and generate sign-in and sign-out URLs. An app can also specify access restrictions for URL paths based on Google Accounts, using the deployment descriptor.

The <security-constraint> element defines a security constraint for URLs that match a pattern. If a user accesses a URL whose path has a security constraint and the user is not signed in, App Engine redirects the user to the Google Accounts sign-in page. Google Accounts redirects the user back to the application URL after successfully signing in or registering a new account. The app does not need to do anything else to ensure that only signed-in users can access the URL.

A security constraint includes an authorization constraint that specifies which Google Accounts users can access the path. If the authorization constraint specifies a user role of \*, then any users signed in with a Google Account can access the URL. If the constraint specifies a user role of admin, then only registered developers of the application can access the URL. The admin role makes it easy to build administrator-only sections of your site.

    <security-constraint>  
        <web-resource-collection>  
            <web-resource-name>profile</web-resource-name>  
            <url-pattern>/profile/\*</url-pattern>  
        </web-resource-collection>  
        <auth-constraint>  
            <role-name>\*</role-name>  
        </auth-constraint>  
    </security-constraint>  
  
    <security-constraint>  
        <web-resource-collection>  
            <web-resource-name>admin</web-resource-name>  
            <url-pattern>/admin/\*</url-pattern>  
        </web-resource-collection>  
        <auth-constraint>  
            <role-name>admin</role-name>  
        </auth-constraint>  
    </security-constraint>

**Note:** G Suite domain administrators and App Engine domain administrators are not included in the **admin** role in this context. Only the application developers, such as those in **Viewer**, **Owner**, or **Developer** roles, can access these portions of the application.

App Engine does not support custom security roles (<security-role>) or alternate authentication mechanisms (<login-config>) in the deployment descriptor.

Security constraints apply to static files as well as servlets.

Secure URLs

Google App Engine supports secure connections via HTTPS for URLs using the [***REGION\_ID***](https://cloud.google.com/appengine/docs/standard/java/config/webxml#appengine-urls).r.appspot.com domain. When a request accesses a URL using HTTPS, and that URL is configured to use HTTPS in the web.xml file, both the request data and the response data are encrypted by the sender before they are transmitted, and decrypted by the recipient after they are received. Secure connections are useful for protecting customer data, such as contact information, passwords, and private messages.

**Note:** To use G Suite domains with HTTPS, you must first [activate and configure SSL for App Engine with your domain](https://cloud.google.com/appengine/docs/standard/java/securing-custom-domains-with-ssl). Otherwise, users attempting to view pages via HTTPS on your domain will likely see timeouts, errors, or warnings.

To declare that HTTPS should be used for a URL, you set up a security constraint in the deployment descriptor (as described in [Security and authentication](https://cloud.google.com/appengine/docs/standard/java/config/webxml#Security_and_Authentication)) with a <user-data-constraint> whose <transport-guarantee> is CONFIDENTIAL. For example:

    <security-constraint>  
        <web-resource-collection>  
            <web-resource-name>profile</web-resource-name>  
            <url-pattern>/profile/\*</url-pattern>  
        </web-resource-collection>  
        <user-data-constraint>  
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>  
        </user-data-constraint>  
    </security-constraint>

Requests using HTTP (non-secure) for URLs whose transport guarantee is CONFIDENTIAL are automatically redirected to the same URL using HTTPS.

Any URL can use the CONFIDENTIAL transport guarantee, including JSPs and static files.

The development web server does not support HTTPS connections. It ignores the transport guarantee, so paths intended for use with HTTPS can be tested using regular HTTP connections to the development web server.

When you test your app's HTTPS handlers using the versioned appspot.com URL, such as https://1.latest.***your\_app\_id***.[***REGION\_ID***](https://cloud.google.com/appengine/docs/standard/java/config/webxml#appengine-urls).r.appspot.com/, your browser warns you that the HTTPS certificate was not signed for that specific domain path. If you accept the certificate for that domain, pages will load successfully. Users will not see the certificate warning when accessing https://***your\_app\_id***.***REGION\_ID***.r.appspot.com/.

You can also use an alternate form of the versioned appspot.com URL designed to avoid this problem by replacing the periods separating the subdomain components with the string "-dot-". For instance, the previous example could be accessed without a certificate warning at https://***VERSION\_ID***-dot-default-dot-***PROJECT\_ID***.[***REGION\_ID***](https://cloud.google.com/appengine/docs/standard/java/config/webxml#appengine-urls).r.appspot.com.

Google Accounts sign-in and sign-out are always performed using a secure connection and is unrelated to how the application's URLs are configured.

As mentioned above, security constraints apply to static files as well as servlets. This includes the transport guarantee.

**Note:** Google [recommends](http://googleonlinesecurity.blogspot.com/2014/08/https-as-ranking-signal_6.html)using the HTTPS protocol to send requests to your app. Google does not issue SSL certificates for double-wildcard domains hosted at **appspot.com**. Therefore with HTTPS you must use the string "-dot-" instead of "." to separate subdomains, as shown in the examples below. You can use a simple "." with your own custom domain or with HTTP addresses.

The welcome file list

When the URLs for your site represent paths to static files or JSPs in your WAR, it is often a good idea for paths to directories to do something useful as well. A user visiting the URL path /help/accounts/password.jsp for information on account passwords might try to visit /help/accounts/ to find a page introducing the account system documentation. The deployment descriptor can specify a list of filenames that the server should try when the user accesses a path that represents a WAR subdirectory that is not already explicitly mapped to a servlet. The servlet standard calls this the "welcome file list."

For example, if the user accesses the URL path /help/accounts/, the following <welcome-file-list> element in the deployment descriptor tells the server to check for help/accounts/index.jsp and help/accounts/index.html before reporting that the URL does not exist:

    <welcome-file-list>  
        <welcome-file>index.jsp</welcome-file>  
        <welcome-file>index.html</welcome-file>  
    </welcome-file-list>

Filters

A *filter* is a class that acts on a request like a servlet, but can allow the handling of the request to continue with other filters or servlets. A filter may perform an auxiliary task, such as logging, performing specialized authentication checks, or annotating the request or response objects before calling the servlet. Filters allow you to compose request processing tasks from the deployment descriptor.

A filter class implements the javax.servlet.Filter interface, including the doFilter() method. Here is a simple filter implementation that logs a message, and passes control down the chain, which may include other filters or a servlet, as described by the deployment descriptor:

package mysite.server;  
  
import java.io.IOException;  
import java.util.logging.Logger;  
import javax.servlet.Filter;  
import javax.servlet.FilterChain;  
import javax.servlet.FilterConfig;  
import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
  
public class LogFilterImpl implements Filter {  
  
    private FilterConfig filterConfig;  
    private static final Logger log = Logger.getLogger(LogFilterImpl.class.getName());  
  
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)  
        throws IOException, ServletException {  
        log.warning("Log filter processed a " + getFilterConfig().getInitParameter("logType")  
            + " request");  
  
        filterChain.doFilter(request, response);  
    }  
  
    public FilterConfig getFilterConfig() {  
        return filterConfig;  
    }  
  
    public void init(FilterConfig filterConfig) {  
        this.filterConfig = filterConfig;  
    }  
  
    public void destroy() {}  
  
}

Similar to servlets, you configure a filter in the deployment descriptor by declaring the filter with the <filter> element, then mapping it to a URL pattern with the <filter-mapping> element. You can also map filters directly to other servlets.

The <filter> element contains a <filter-name>, <filter-class>, and optional <init-param> elements.

    <filter>  
        <filter-name>logSpecial</filter-name>  
        <filter-class>mysite.server.LogFilterImpl</filter-class>  
        <init-param>  
            <param-name>logType</param-name>  
            <param-value>special</param-value>  
        </init-param>  
    </filter>

The <filter-mapping> element contains a <filter-name> that matches the name of a declared filter, and either a <url-pattern> element for applying the filter to URLs, or a <servlet-name> element that matches the name of a declared servlet for applying the filter whenever the servlet is called.

    <!-- Log for all URLs ending in ".special" -->  
    <filter-mapping>  
        <filter-name>logSpecial</filter-name>  
        <url-pattern>\*.special</url-pattern>  
    </filter-mapping>  
  
    <!-- Log for all URLs that use the "comingsoon" servlet -->  
    <filter-mapping>  
        <filter-name>logSpecial</filter-name>  
        <servlet-name>comingsoon</servlet-name>  
    </filter-mapping>

**Note:** Filters are not invoked on static assets, even if the path matches a **filter-mapping** pattern. Static files are served directly to the browser.

Error Handlers

You can customize what the server sends to the user when an error occurs, using the deployment descriptor. The server can display an alternate page location when it's about to send a particular HTTP status code, or when a servlet raises a particular Java exception.

The <error-page> element contains either an <error-code> element with an HTTP error code value (such as 500), or an <exception-type> element with the class name of the expected exception (such as java.io.IOException). It also contains a <location> element containing the URL path of the resource to show when the error occurs.

    <error-page>  
        <error-code>500</error-code>  
        <location>/errors/servererror.jsp</location>  
    </error-page>

**Note:** At present, you cannot configure custom error handlers for some error conditions. Specifically, you cannot customize the HTTP **404** response page when no servlet mapping is defined for a URL, the **403** quota error page, or the **500** server error page that appears after an App Engine internal error.

Unsupported web.xml features

The following web.xml features are not supported by App Engine:

* App Engine supports the <load-on-startup> element for servlet declarations. However, the load actually occurs during the first request handled by the web server instance, not prior to it.
* Some deployment descriptor elements can take a human readable display name, description and icon for use in IDEs. App Engine doesn't use these, and ignores them.
* App Engine doesn't support JNDI environment variables (<env-entry>).
* App Engine doesn't support EJB resources (<resource-ref>).
* Notification of the destruction of servlets, servlet context, or filters is not supported.
* The <distributable> element is ignored.
* Servlet scheduling with <run-at> is not supported.

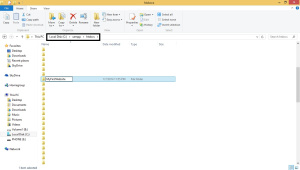
6.CREATING WEBSITES USING PHP

Website using PHP

**1.) Setting up your server**

Now that your all set and go, as the basics of programming goes, let's start by creating a simple shoutout of *"hello world"* in our server.

First of, go to the directory where you installed your XAMPP (Commonly in *C:\xampp*). From there go to the *htdocs* folder (Commonly in *C:\xampp\htdocs*) and create a folder named "*MyFirstWebsite*".

[](http://xtiology.files.wordpress.com/2014/01/1-php101.jpg)

From that part, you have now created a Local URL for your website. That folder will be used to supply in all website files (.html, .php, .css, .js, etc.). Open up your text editor and let's get started!

*I use sublime text as my text editor. If your using Notepad++ or any others, it's ok. It's not really a big factor but it's just a preference on which one would you like to use.*

What we will do is a basic HTML page and display "hello world" from the server using a basic PHP syntax. We then type the following syntax:

Hide   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<body>

<?php

echo "<p>Hello World!</p>";

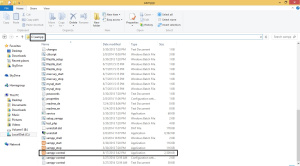
?>

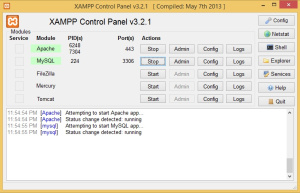
</body>

</html>

Save the file to the "*MyFirstWebSite*" Folder and name it as "*index.php*". (Directory as seen on the top part of the image)

Now that you have the file. Let's now open your XAMPP control panel. In case it doesn't appear on your desktop, it is located in your XAMPP folder as seen on the image:

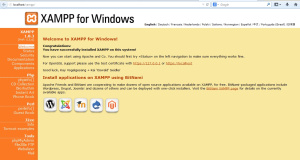
[](http://xtiology.files.wordpress.com/2014/03/untitled75.jpg)

[](http://xtiology.files.wordpress.com/2014/03/untitled2.jpg)

Now that it's there, Run your Apache and mySQL by clicking the "Start" button on the actions column. You should see a random PID(s) and the default port number. **Apache** is the name of our web server wherein it will handle all the files as well as serve as the communication to the web browser and **MySQL** is our database which will store all of our information.

Open up your web browser and in the address bar, type **localhost**. You should see the menu of your XAMPP.

*If it's the first time you run it, it will ask what language would you prefer, just simply pick and it will lead you to the menu. If you will noticed the directory is localhost/xampp, it's where the default page leads you even if you type in localhost.*

[](http://xtiology.files.wordpress.com/2014/03/untitled3.jpg)

If that appears, then it means that your XAMPP server is now running. Now let's try running the website you just placed. Try typing **localhost/MyFirstWebsite.**It should be the same as the picture below.

[Untitled](http://xtiology.files.wordpress.com/2014/03/untitled4.jpg)

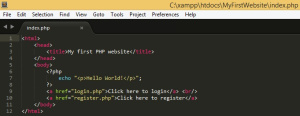
If you will notice that the URL is *MyFirstWebsite*, it is derived from the htdocs folder and it automatically reads files that are named "index"(Be it index.html, index.aspx, etc), which serve as the default page. Typing **localhost/MyfirstWebsite/index.php**is just the same. You can also create your custom name for the URL by simply renaming the folder but let's just stick to *MyFirstWebsite*.

*Note: If you don't have a file named index and you entered the URL, you will receive an error 404 for not having the file on the server. If you do have different files that are not named index.<extention>, you have to specify the specific file name. E.x:****localhost/MyfirstWebsite****/****page.php***

**2.) Creating the public HTML Pages**

Next, let's re-modify our website and add a registration link where our users can register as well as a Log-in page right after getting registered. Let's modify our home page with the following code:

***index.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled5.jpg)

Hide   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<body>

<?php

echo "<p>Hello World!</p>";

?>

<a href="login.php"> Click here to login

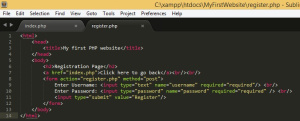
<a href="register.php"> Click here to register

</body>

</html>

As you can see, we only added 2 links which are for the Login and register. Let's create the registration page first:

***register.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled6.jpg)

Hide   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<body>

<h2>Registration Page</h2>

<a href="index.php">Click here to go back<br/><br/>

<form action="register.php" method="POST">

Enter Username: <input type="text" name="username" required="required" /> <br/>

Enter password: <input type="password" name="password" required="required" /> <br/>

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

</form>

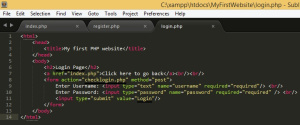
</body>

</html>

As you can see, it's just a basic form where the user can input his/her credentials. Now for the login page:

*Hint: Just copy-paste the same code to make things faster.*

***login.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled7.jpg)

Hide   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<body>

<h2>Login Page</h2>

<a href="index.php">Click here to go back<br/><br/>

<form action="checklogin.php" method="POST">

Enter Username: <input type="text" name="username" required="required" /> <br/>

Enter password: <input type="password" name="password" required="required" /> <br/>

<input type="submit" value="Login"/>

</form>

</body>

</html>

Basically it's just the same code as from the *register.php* but the changes made were the ones underlined.

*Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/login.php)*for the complete login.php code*

Try running ***localhost/MyFirstWebsite*** again and your pages should look like this:

**index.php**

[Untitled](http://xtiology.files.wordpress.com/2014/03/untitled8.jpg)

**login.php**

[Untitled](http://xtiology.files.wordpress.com/2014/03/untitled73.jpg)

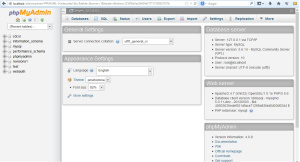
**register.php**

[Untitled](http://xtiology.files.wordpress.com/2014/03/untitled74.jpg)

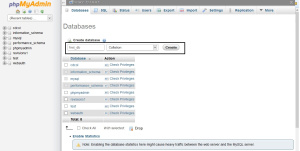
**3.) Creating the database and it's tables**

Now that we have our basic page for the public. Let's proceed to the database. First, type **localhost/phpmyadmin**. This will lead you to the phpmyadmin homepage:

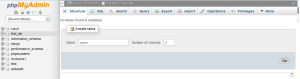
***localhost/phpmyadmin***

[](http://xtiology.files.wordpress.com/2014/03/untitled12.jpg)

From there, go to the Databases tab located on top then from the text box in the middle, type *first\_db* then click on create. Just leave the *Collation* as is.

[](http://xtiology.files.wordpress.com/2014/03/untitled13.jpg)

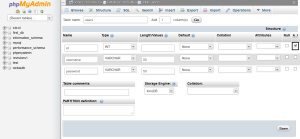
You have just successfully created your first database. Now from there, let's create a table wherein we can register our users and display information. First, click on*first\_db* located on the left side and create a table named *users* with 3 columns then click on *Go*.

[](http://xtiology.files.wordpress.com/2014/03/untitled15.jpg)

For the table's structure, make sure to have the following fields then click on save:

* Format: Column Name - Type - Length - Null Property - Other Properties
* id - INT - N/A - Not Null - Auto Increment
* username - varchar - 50 - Not null
* password - varchar - 50 - Not null

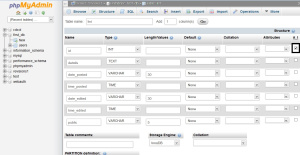
*Leave everything by default if not specified.*

[](http://xtiology.files.wordpress.com/2014/03/untitled16.jpg)

*Note: You need to scroll right for the auto\_increment. I just edited the picture to fit the A\_I field*

Next, create another table named *list* with 7 columns and for the table's structure:

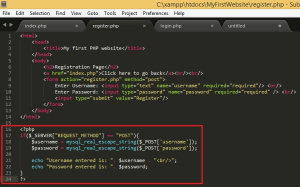
* id - INT - N/A - Not Null - Auto Increment
* details - text - Not null
* date\_posted - varchar - 30 - Not null
* time\_posted - Time - Not null
* date\_edited - varchar - 30 - Not null
* time\_edited - Time - Not null
* public -varchar - 5 - Not null

[](http://xtiology.files.wordpress.com/2014/03/untitled34.jpg)

**4.)Adding users to the database**

Now that we have our tables. Let's move on to the fun part, getting your registration page into action. From your *registration.php*, add the following below the html codes:

***register.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled18.jpg)

Hide   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<body>

<h2>Registration Page</h2>

<a href="index.php">Click here to go back<br/><br/>

<form action="checklogin.php" method="POST">

Enter Username: <input type="text" name="username" required="required" /> <br/>

Enter password: <input type="password" name="password" required="required" /> <br/>

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

</form>

</body>

</html>

Here's the explanation to the code:

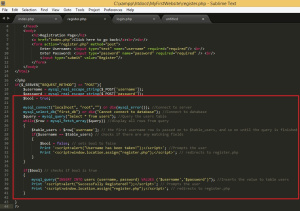
* ***$\_SERVER["REQUEST\_METHOD"] == "POST"*** - checks if the form has received a POST method when the *submit*button has been clicked. The *POST* method is created in the html from the ***method="POST"****.* Click [here](http://www.w3schools.com/tags/att_form_method.asp) for the form method reference.
* ***$\_POST['']*** - gets the name coming from a POST method. This just simply gets the input based on the name from the form. In our case it's username and password.
* **mysql\_real\_escape\_string()** - encapsulates the input into a string to prevent inputs from SQL Injections. This ensures that your strings don't escape from unnecessary characters. Click [here](http://technet.microsoft.com/en-us/library/ms161953%28v=sql.105%29.aspx) to learn more about SQL Injections.

Now try to go to your *register.php* and try to input anything then click on "Register". In my case I placed in the username *xtian* and password as *123456*. It should display the inputs below. Here's my sample:

[](http://xtiology.files.wordpress.com/2014/03/untitled19.jpg)

On this part, you should have understood on how to get input from the form. Now to add it to the database. On your *register.php*, add the following code:

***register.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled20.jpg)

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<body>

<h2>Registration Page</h2>

<a href="index.php">Click here to go back<br/><br/>

<form action="checklogin.php" method="POST">

Enter Username: <input type="text" name="username" required="required" /> <br/>

Enter password: <input type="password" name="password" required="required" /> <br/>

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

</form>

</body>

</html>

alert("Username has been taken!");</script>'; // Prompts the user

Print '<script>window.location.assign("register.php");</script>'; // redirects to register.php

}

}

if($bool)

{

mysql\_query("INSERT INTO users (username, password) VALUES ('$username', 'password')"); // inserts value into table users

Print '<script>alert("Successfully Registered!");</script>'; // Prompts the user

Print '<script>window.location.assign("register.php");</script>'; // redirects to register.php

}

}

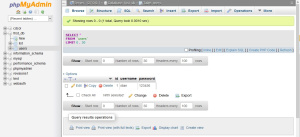
?>

*Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/register.php)*for the complete register.php file*

Here are the explanations to the code:

* ***mysql\_connect("Server name","Server Username","Server Password")*** - The syntax used to connect to our XAMPP server. ***localhost*** or ***127.0.0.1*** is the name of the server. The default username is *root* and no password for default.
* ***mysql\_select\_db("database name")*** - Selects the database to be used.
* ***or die('Message')*** - Displays the error message if the condition wasn't met.
* ***mysql\_query('sql query')*** - does the SQL queries. Click [here](http://www.w3schools.com/sql/default.asp) for some SQL query samples. The again, i'm not here to discuss about mySQL.
* ***mysql\_fetch\_array('query')*** - fetches all queries in the table to display or manipulate data. It is placed in a while loop so that it would query all rows. *Take note that only 1 row is queried per loop that's why a while loop is necessary.*
* ***$row['row name']*** - the value of the column in the current query. It is represented as an array. In our case $row is the name of the variable for our row in the loop.

Try the inputs that you have made earlier and see what happens. It should prompt that you have successfully registered. Try going to *phpmyadmin* and see your *users table:*

[](http://xtiology.files.wordpress.com/2014/03/untitled21.jpg)

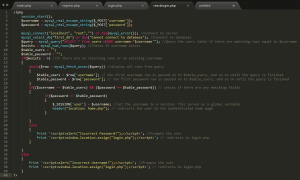
Congratulations! Now you know how to add data into the database with data validations.

**5.)User log-in: Authentication**

Now for the login page. Let's create a new file called*checklogin.php.*The reason is going back to our*login.php,*our form has an action called *"checklogin.php",*particularly <form ***action = "checklogin.php"*** method= "POST">. If you will notice on the *register.php*, it's also on *register.php* because the back-end is done on the same file as well.

Let's now code the *checklogin.php* with the following syntax:

***checklogin.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled23.jpg)

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<?php

session\_start();

$username = mysql\_real\_escape\_string($\_POST['username']);

$password = mysql\_real\_escape\_string($\_POST['password']);

$bool = true;

mysql\_connect("localhost", "root", "") or die (mysql\_error()); *//Connect to server*

mysql\_select\_db("first\_db") or die ("Cannot connect to database"); *//Connect to database*

$query = mysql\_query("Select \* from users WHERE username='$username'"); *// Query the users table*

$exists = mysql\_num\_rows($query); *//Checks if username exists*

$table\_users = "":

$table\_password = "";

if($exists > 0) *//IF there are no returning rows or no existing username*

{

while($row = mysql\_fetch\_assoc($query)) *// display all rows from query*

{

$table\_users = $row['username']; *// the first username row is passed on to $table\_users, and so on until the query is finished*

$table\_password = $row['password']; *// the first password row is passed on to $table\_password, and so on until the query is finished*

}

if(($username == $table\_users) && ($password == $table\_password))*// checks if there are any matching fields*

{

if($password == $table\_password)

{

$\_SESSION['user'] = $username; *//set the username in a session. This serves as a global variable*

header("location: home.php"); *// redirects the user to the authenticated home page*

}

}

else

{

Print '<script>alert("Incorrect Password!");</script>'; *// Prompts the user*

Print '<script>window.location.assign("login.php");</script>'; *// redirects to login.php*

}

}

else

{

Print '<script>alert("Incorrect username!");</script>'; *// Prompts the user*

Print '<script>window.location.assign("login.php");</script>'; *// redirects to login.php*

}

?>

*Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/checklogin.php)*for the complete checklogin.php code*

Here's the explanation of the code (Some are explained earlier so there's no need to reiterate):

* ***session\_start()*** - Starts the session. This is usually done on authenticated pages. The reason why we used this is because this is required for the *$\_SESSION['']*.
* ***mysql\_num\_rows()*** - This returns an integer. This counts all the rows depending on the query.
* ***$\_SESSION['name']*** - Serves as the session name for the entire session. This is relatively similar to public variables in object-oriented programming. We will be using this for validating whether the user is authenticated or not.

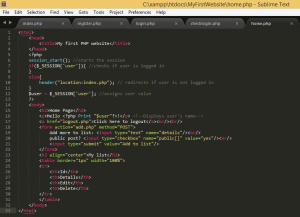
Now try to test your input with a wrong username and password. It should return the desired prompt. After testing, try inputting the correct values. It should lead you to *home.php*.

***Note: home.php does not exist yet so it will produce an error 404.***

**6.)Setting up the home page for Logged-in users and Logging-out**

Now that were authenticated, let now create our home page (*home.php*) with this following syntax:

***home.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled31.jpg)

Hide   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<?php

session\_start(); *//starts the session*

if($\_SESSION['user']){ *// checks if the user is logged in*

}

else{

header("location: index.php"); *// redirects if user is not logged in*

}

$user = $\_SESSION['user']; *//assigns user value*

?>

<body>

<h2>Home Page</h2>

<hello>!

<!--*Display's user name-->*

<a href="logout.php">Click here to go logout</a><br/><br/>

<form action="add.php" method="POST">

Add more to list: <input type="text" name="details" /> <br/>

Public post? <input type="checkbox" name="public[]" value="yes" /> <br/>

<input type="submit" value="Add to list"/>

</form>

<h2 align="center">My list</h2>

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Details** | **Edit** | **Delete** |

</body> </html>

Here's the explanation to the code:

* ***session\_start()*** - Basically starts the session. Required for $\_SESSION[''].
* ***header()*** - redirects the user.

Try refreshing your browser and it should look like this:

[](http://xtiology.files.wordpress.com/2014/03/untitled32.jpg)

Now that we have our home, let's try creating our *logout.php*and test if the user's session is off. What we would do is that if the user is logged-out, the user shouldn't access *home.php.* So here's the simple syntax to *logout.php*:

***logout.php***

[Untitled](http://xtiology.files.wordpress.com/2014/03/untitled26.jpg)

Hide   Copy Code

<?php

session\_start();

session\_destroy();

header("location:index.php");

?>

The syntax is simple. ***session\_destroy()*** simply remove's all session's meaning, the value of ***$\_SESSION['']*** will be removed and ***header()*** will simply redirect it to the home page.

**7.)Testing Page Security**

Now try refreshing home.php and click on logout. Now try Clicking on the back button of your browser and see what happens:

[Untitled](http://xtiology.files.wordpress.com/2014/03/untitled27.jpg)

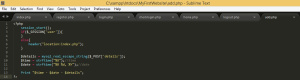
As you can see, it doesn't direct you to*home.php*because you are logged-out. Now for the second test, try manually inputting the address ***localhost/MyFirstWebsite/home.php***. The same case should also happen. Since were logged-out, even a manual input of the address doesn't access an authorized page. What we have done is a simple security mechanism wherein we redirect back unauthorized users into a public page.

Now try logging in again and let's go back to *home.php*.

**8.) Adding data to the list - User Access Only**

In our next step, let's create the adding of item's to the list. As you will notice from the form, it is written as <form **action="add.php"** method="POST">, meaning our http post request goes to *add.php*and with that, we create our *add.php* with the following syntax:

***add.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled28.jpg)

Hide   Copy Code

<?php

session\_start();

if($\_SESSION['user']){

}

else{

header("location:index.php");

}

$details = mysql\_real\_escape\_string($\_POST['details']);

$time = strftime("%X"); *//time*

$date = strftime("%B %d, %Y"); *//date*

Print "$time - $date - $details";

?>

Take note that this ain't our official *add.php* syntax yet, I'm just going to demonstrate the time and date syntax and getting your input.

Now go back to your *home.php* and try to add an item then click on "Add to list".

[](http://xtiology.files.wordpress.com/2014/03/untitled29.jpg)

This should be the following output on *add.php*:

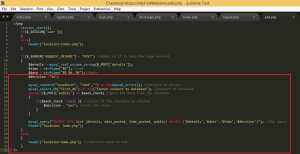
[Untitled](http://xtiology.files.wordpress.com/2014/03/untitled35.jpg)

As you can see, we have our current time, date, and your input. Here's the explanation to the code:

* ***strftime()*** - get's the time based on what format your placed.
* ***%X*** - current system time.
* ***%B*** - current system month.
* ***%d*** - current system day.
* ***%Y*** - current system year.

Now let's modify our *add.php* and add the following data into the database together with the data from the checkbox:

***add.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled36.jpg)

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<?php

session\_start();

if($\_SESSION['user']){

}

else{

header("location:index.php");

}

if($\_SERVER['REQUEST\_METHOD'] == "POST")

{

$details = mysql\_real\_escape\_string($\_POST['details']);

$time = strftime("%X"); *//time*

$date = strftime("%B %d, %Y"); *//date*

$decision = "no";

mysql\_connect("localhost","root","") or die(mysql\_error()); *//Connect to server*

mysql\_select\_db("first\_db") or die("Cannot connect to database"); *//Conect to database*

foreach($\_POST['public'] in $each\_check) *//gets the data from the checkbox*

{

if($each\_check != null){ *//checks if checkbox is checked*

$decision = "yes"; *// sets the value*

}

}

mysql\_query("INSERT INTO list(details, date\_posted, time\_posted, public) VALUES ('$details','$date','$time','$decision')"); *//SQL query*

header("location:home.php");

}

else

{

header("location:home.php");

}

?>

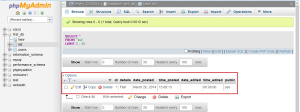
*Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/add.php)*for the complete add.php code*

Here's a little explanation:

* **foreach()** - gets the value of the checkbox. As you will notice, the checkbox format in the form is *name="checkbox[]"*. To get data from checkbox, it has to be instantiated as an array. Doing so would make it possible to get data from multiple checkboxes.

Now try entering some data and click "Add to list". In my case, I'll just use fish again. Let's go to our phpmyadmin and let's see if the data has been added. Here's the result of my case:

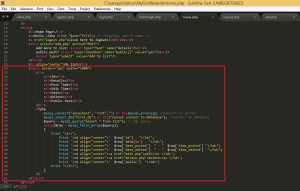
***localhost/phpmyadmin***

[](http://xtiology.files.wordpress.com/2014/03/untitled37.jpg)

**9.)Displaying data in the home page**

Now that we have seen that the data has been successfully added. Let's now display the data in our home page. Let's modify our *home.php* and let's add some columns for the date:

***home.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled40.jpg)

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<?php

session\_start(); *//starts the session*

if($\_SESSION['user']){ *// checks if the user is logged in*

}

else{

header("location: index.php"); *// redirects if user is not logged in*

}

$user = $\_SESSION['user']; *//assigns user value*

?>

<body>

<h2>Home Page</h2>

<hello>!

<!--*Display's user name-->*

<a href="logout.php">Click here to go logout</a><br/><br/>

<form action="add.php" method="POST">

Add more to list: <input type="text" name="details" /> <br/>

Public post? <input type="checkbox" name="public[]" value="yes" /> <br/>

<input type="submit" value="Add to list"/>

</form>

<h2 align="center">My list</h2>

';

Print '";

Print '";

Print '";

Print '";

Print '';

Print '';

Print '';

Print '';

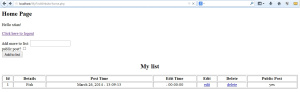
}

?>

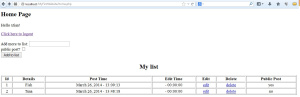
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Id** | **Details** | **Post Time** | **Edit Time** | **Edit** | **Delete** | **Public Post** |
| '. $row['id'] . " | '. $row['details'] . " | '. $row['date\_posted'] . " - " . $row['time\_posted'] . " | '. $row['date\_edited'] . " - " . $row['time\_edited'] ." | [edit](https://www.codeproject.com/Articles/759094/edit.php) | [delete](https://www.codeproject.com/Articles/759094/delete.php) | '. $row['public'] . ' |

</body> </html>

The explanation to the added code is quite simple. It just basically displays the data coming from the while loop. It has been explained earlier in our tutorial so I believe that by this point, you should have understood the process of getting the data in the query. Going back to the browser, try refreshing your *home.php* and see what happens:

[](http://xtiology.files.wordpress.com/2014/03/untitled41.jpg)

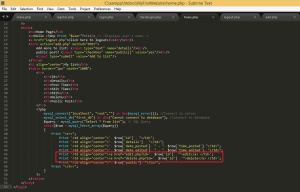
It should now display that data. From our CRUD checklist, we have now accomplished Create and Read. Next is to update(edit) and delete information. If you will notice we have edit and delete links displayed on the column. I'll add another data to the list named "*tuna*" to have another example and this time, it's privacy to *no*:

[](http://xtiology.files.wordpress.com/2014/03/untitled42.jpg)

**10.) Editing Data**

Let's now try editing our data and to do that we will use a new functionality called "*GET*". With our previous methods, we have been using *POST*as our http request but this time, let's use *GET*for editing and deleting records. To start of, let's modify our *home.php*and add a little code to 2 columns:

***home.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled43.jpg)

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<?php

session\_start(); *//starts the session*

if($\_SESSION['user']){ *// checks if the user is logged in*

}

else{

header("location: index.php"); *// redirects if user is not logged in*

}

$user = $\_SESSION['user']; *//assigns user value*

?>

<body>

<h2>Home Page</h2>

<hello>!

<!--*Display's user name-->*

<a href="logout.php">Click here to go logout</a><br/><br/>

<form action="add.php" method="POST">

Add more to list: <input type="text" name="details" /> <br/>

Public post? <input type="checkbox" name="public[]" value="yes" /> <br/>

<input type="submit" value="Add to list"/>

</form>

<h2 align="center">My list</h2>

';

Print '";

Print '";

Print '";

Print '";

Print '';

Print '';

Print '';

Print '';

}

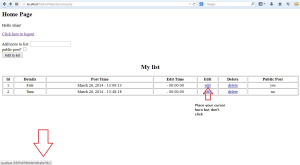
?>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Id** | **Details** | **Post Time** | **Edit Time** | **Edit** | **Delete** | **Public Post** |
| '. $row['id'] . " | '. $row['details'] . " | '. $row['date\_posted'] . " - " . $row['time\_posted'] . " | '. $row['date\_edited'] . " - " . $row['time\_edited'] ." | [edit](https://www.codeproject.com/Articles/759094/edit.php?id=%27.%20$row%5b%27id%27%5d%20.%27) | [delete](https://www.codeproject.com/Articles/759094/delete.php?id=%27.%20$row%5b%27id%27%5d%20.%27) | '. $row['public'] . ' |

</body> </html>

If you have noticed, we only added URL parameter for the edit and delete links namely *id.*We will be using this later to handle thee data. The reason why we use *id*is because it's a unique identifier. It is possible for the person to have enter the same data so it's not recommended to use the *details*as a mean for manipulation later on.

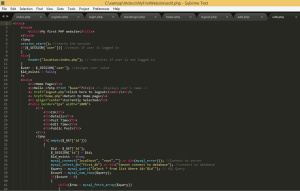
Try putting your cursor into the edit link and you will see the value of the id onthe lower left:

[](http://xtiology.files.wordpress.com/2014/03/untitled44.jpg)

Now that we have that, let's try creating our *edit.php* and let's try to get the data from there:

***edit.php***

lines 1 - 42

[](http://xtiology.files.wordpress.com/2014/03/untitled56.jpg)

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<?php

session\_start(); *//starts the session*

if($\_SESSION['user']){ *// checks if the user is logged in*

}

else{

header("location: index.php"); *// redirects if user is not logged in*

}

$user = $\_SESSION['user']; *//assigns user value*

?>

<body>

<h2>Home Page</h2>

<hello>!

<!--*Display's user name-->*

<a href="logout.php">Click here to go logout</a><br/><br/>

<a href="home.php">Return to home page</a>

<h2 align="center">Currently Selected</h2>

0)

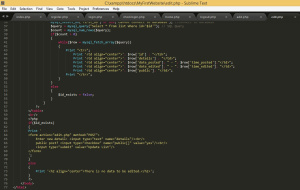
{

while($row = mysql\_fetch\_array($query))

{

Some explanations to the code:

* ***!empty()*** - a method that checks if the value is not empty. The syntax can be reversed if you want to check if it's empty by removing the explanation point (!), therefore it's syntax would be ***empty()***.
* ***$\_GET['']*** - Used to get the value from the parameter. In our case, we use *id*as our URL parameter so the syntax would be $\_GET['id'].
* ***$id\_exists*** - the variable that checks whether the given id exists.
* ***$\_SESSION['id']*** - we place the value of id into session to use it on another file.

lines 42 - 76 [](http://xtiology.files.wordpress.com/2014/03/untitled57.jpg)

Hide   Copy Code

Print "<tr>";

Print '

"; Print '"; Print '"; Print '"; Print '"; Print ""; } } else { $id\_exists = false; } } ?>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Details** | **Post Time** | **Edit Time** | **Public Post** |
| ' . $row['id'] . " | ' . $row['details'] . " | ' . $row['date\_posted'] . " - " . $row['time\_posted']." | ' . $row['date\_edited'] . " - " . $row['time\_edited']." | ' . $row['public'] . " |

<br/> Enter new detail: <input type="text" name="details"/> <br/> public post? <input type="checkbox name="public[]" value="yes"/> <input type="submit" value="Update List"/> </form> '; } else { <h2 align="center">There is not data to be edited.</h2> } ?> </body> </html> *Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/edit.php)*for the complete edit.php code (Only refer to the front-end part. Github Sourced)*

The reason why we are putting the variable *$id\_exists*is because in case the user modifies the URL parameter into a non-existing number(in our case we only have 2 rows/id's), we can display a prompt in which the data doesn't exist.

Now try clicking the edit link into the first row and it should display like this:

[](http://xtiology.files.wordpress.com/2014/03/untitled49.jpg)

Complete edit.php code:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<html>

<head>

<title>My first PHP Website</title>

</head>

<?php

session\_start(); *//starts the session*

if($\_SESSION['user']){ *// checks if the user is logged in*

}

else{

header("location: index.php"); *// redirects if user is not logged in*

}

$user = $\_SESSION['user']; *//assigns user value*

?>

<body>

<h2>Home Page</h2>

<hello>!

<!--*Display's user name-->*

<a href="logout.php">Click here to go logout</a><br/><br/>

<a href="home.php">Return to home page</a>

<h2 align="center">Currently Selected</h2>

0)

{

while($row = mysql\_fetch\_array($query))

{

Print "";

Print '";

Print '";

Print '";

Print '";

Print '";

Print "";

}

}

else

{

$id\_exists = false;

}

}

?>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Details** | **Post Time** | **Edit Time** | **Public Post** |
| ' . $row['id'] . " | ' . $row['details'] . " | ' . $row['date\_posted'] . " - " . $row['time\_posted']." | ' . $row['date\_edited'] . " - " . $row['time\_edited']." | ' . $row['public'] . " |

<br/> Enter new detail: <input type="text" name="details"/> <br/> public post? <input type="checkbox name="public[]" value="yes"/> <input type="submit" value="Update List"/> </form> '; } else { <h2 align="center">There is not data to be edited.</h2> } ?> </body> </html> *Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/edit.php)*for the complete edit.php code (Only refer to the front-end code. Github sourced)*

Let's try modifying the URL parameter by removing ***?id=1***and now should result to ***localhost/MyFirstWebsite/edit.php*** and it should result like this:

[](http://xtiology.files.wordpress.com/2014/03/untitled53.jpg)

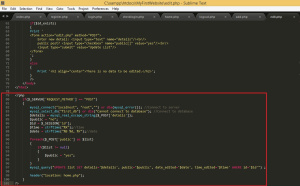
Now try putting a value that is greater than the id number, in our case, let's try 5 and it should result like this:

***localhost/MyFirstWebsite/edit.php?id=5***

[](http://xtiology.files.wordpress.com/2014/03/untitled55.jpg)

Now that we secured our URL parameters, lets now place the edit syntax. Let's go back to *edit.php* and add some following code to update the information to the database:

***edit.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled58.jpg)

Hide   Copy Code

<?php

if($\_SERVER['REQUEST\_METHOD'] == "POST")

{

mysql\_connect("localhost", "root", "") or die (mysql\_error()); *//Connect to server*

mysql\_select\_db("first\_db") or die ("Cannot connect to database"); *//Connect to database*

$details = mysql\_real\_escape\_string($\_POST['details']);

$public = "no";

$id = $\_SESSION['id'];

$time = strftime("%X"); *//time*

$date = strftime("%B %D, %Y"); *//date*

foreach($\_POST['public'] as $list)

{

if($list != null)

{

$public = "yes";

}

}

mysql\_query("UPDATE list SET details='$details', public='$public', date\_edited='$date', time\_edited='$time' WHERE id='$id'");

header("location:home.php");

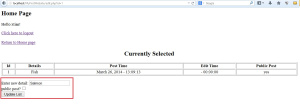
}

?>

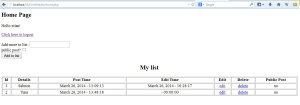
*Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/edit.php)*for the complete edit.php code (Github)*

Now try refreshing and go back to the edit page. Let's try a different data. In my case, I'll be using "Salmon" and change it to non-public:

***edit.php?id=1***

[](http://xtiology.files.wordpress.com/2014/03/untitled59.jpg)

Go ahead and click Update list and you should be redirected to *home.php* and see the updated list.

[](http://xtiology.files.wordpress.com/2014/03/untitled60.jpg)

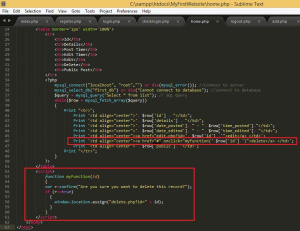
Now, we have a time and date of edit displayed on the *Edit Time* column. Our privacy has been set now to non-public and the value has changed into *Salmon.*

Congratulations! We have now done the edit function!

**11.) Deleting data**

Looking back to the CRUD, we have now done creating(adding), reading(displaying), and updating(editing) records. Now for the last part, deleting records. For this part it's just relatively the same as what we have done on edit but what differs is just the SQL statement. Instead of using UPDATE, we will be using the DELETE syntax. In deleting records, we have to prompt people making sure that they'd really want to delete the record so we will be adding a little javascript in *home.php.* To do that, let's modify our code and add some int *home.php*

***home.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled61.jpg)

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<table border="1px" width="100%">

Id

Details

Post Time

Edit Time

Edit

Delete

Public Post

';

Print ''. $row['id'] . "";

Print ''. $row['details'] . "";

Print ''. $row['date\_posted'] . " - " . $row['time\_posted'] . "";

Print ''. $row['date\_edited'] . " - " . $row['time\_edited'] ."";

Print '<a href="edit.php?id='.$row['id'].'">edit';

Print '<a href="#" nclick="myfunction('.$row['id'].')">delete</a>';

Print ''. $row['public'] . '';

Print '';

}

?>

<script>

function myFunction(id)

{

var r = confirm("Are you sure you want to delete this record?");

if(r == true)

{

window.location.assign("delete.php?id=" + id);

}

}

</script>

*Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/home.php)*for the complete home.php code*

As you have noticed, we edited the link for the delete. We changed *href* into "#" and added and *onclick* function for Javascript for the method of *myFunction*and inside it's parameter is the id of the row. Below the table written is the Javascript syntax wherein it prompts the user if he/she want's to delete the record. If the user confirms, the page then directs to *delete.php* together embedded with the value of the *id*. Now let's create *delete.php* and here's the following syntax:

***delete.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled62.jpg)

Hide   Copy Code

<?php

session\_start(); *//starts the session*

if($\_SESSION['user']){ *//checks if user is logged in*

}

else {

header("location:index.php"); *//redirects if user is not logged in.*

}

if($\_SERVER['REQUEST\_METHOD'] == "GET")

{

mysql\_connect("localhost", "root", "") or die(mysql\_error()); *//connect to server*

mysql\_select\_db("first\_db") or die("cannot connect to database"); *//Connect to database*

$id = $\_GET['id'];

mysql\_query("DELETE FROM list WHERE id='$id'");

header("location:home.php");

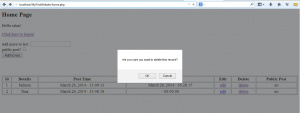
}

?>

*Click*[*here*](https://github.com/xtianxchan/MyFirstWebsite/blob/master/delete.php)*for the complete delete.php code*

The code is just simple and the syntax are also the ones that we used before but noticed we have changed our request method into *GET.*We are now using the *GET* request since we have a URL parameter. Now try refreshing *home.php*and let's try deleting the first record. This should be the result:

***Prompting:***

[](http://xtiology.files.wordpress.com/2014/03/untitled63.jpg)

***End-result:***

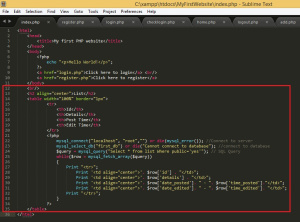
[](http://xtiology.files.wordpress.com/2014/03/untitled64.jpg)

Congratulations! Now we have now officially completed our CRUD statements!

**12.) Displaying public data**

Now for the last part, displaying public data. We will be displaying data that's been set to *yes*in our *index.php*, in which is a page for non-authenticated users. It's very simple. We just have to edit our *index.php*and add a some php code and table. Here's our updated *index.php*:

***index.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled65.jpg)

Now log-out and see your default page. It should look something like this:

[](http://xtiology.files.wordpress.com/2014/03/untitled66.jpg)

*Note: You won't see the data yet since we haven't set any information to public.*

Now let's log-in again and this time, let's add some more data. In my case I've added the following:

* Salad - public
* Corn - non-public
* Pasta - public
* Chicken - public
* Spaghetti - non-public

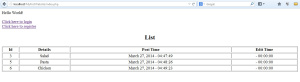
With a total of 6 data's with 3 of each privacy setting:

***home.php***

[](http://xtiology.files.wordpress.com/2014/03/untitled67.jpg)

Now let's log-out and see our default page(*index.php*). It should now look like this:

***index.php***

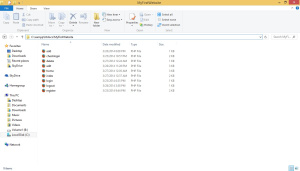
[](http://xtiology.files.wordpress.com/2014/03/untitled69.jpg)

As you can see, it only display's data that are set to public.

Congratulations! We have finally finished the tutorials for this session!

**13.) De-briefing and Summary**

In the end, you should have the following files:

[](http://xtiology.files.wordpress.com/2014/03/untitled71.jpg)

7.UNDERSTANDING JAVASCRIPT

**UNDERSTANDING JAVASCRIPT**

JAVASCRIPT – SYNTAX

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

<script ...>

JavaScript code

</script>

The script tag takes two important attributes:

 **Language:** This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.

 **Type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript syntax will look as follows.

<script language="javascript" type="text/javascript">

JavaScript code

</script>

**Your First JavaScript Code**

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading **15**

the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

<html>

<body>

<script language="javascript" type="text/javascript">

<!--

document.write ("Hello World!")

//-->

</script>

</body>

</html>

This code will produce the following result:

Hello World!

**Whitespace and Line Breaks**

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

**Semicolons are Optional**

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

<script language="javascript" type="text/javascript"> **16**

<!--

var1 = 10

var2 = 20

//-->

</script>

But when formatted in a single line as follows, you must use semicolons:

<script language="javascript" type="text/javascript">

<!--

var1 = 10; var2 = 20;

//-->

</script>

**Note: It is a good programming practice to use semicolons.**

**Case Sensitivity**

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

**NOTE:** Care should be taken while writing variable and function names in JavaScript.

**Comments in JavaScript**

JavaScript supports both C-style and C++-style comments. Thus:

 Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.

 Any text between the characters /\* and \*/ is treated as a comment. This may span multiple lines.

**17**

 JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.

 The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

**Example**

The following example shows how to use comments in JavaScript.

<script language="javascript" type="text/javascript">

<!--

// This is a comment. It is similar to comments in C++

/\*

\* This is a multiline comment in JavaScript

\* It is very similar to comments in C Programming

\*/

//-->

</script> **18**

3. JAVASCRIPT – ENABLING

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera.

**JavaScript in Internet Explorer**

Here are the steps to turn on or turn off JavaScript in Internet Explorer:

 Follow **Tools -> Internet Options** from the menu.

 Select **Security** tab from the dialog box.

 Click the **Custom Level** button.

 Scroll down till you find the **Scripting** option.

 Select *Enable* radio button under **Active scripting**.

 Finally click OK and come out.

To disable JavaScript support in your Internet Explorer, you need to select **Disable** radio button under **Active scripting**.

**JavaScript in Firefox**

Here are the steps to turn on or turn off JavaScript in Firefox:

 Open a new tab -> type **about: config** in the address bar.

 Then you will find the warning dialog. Select **I’ll be careful, I promise!**

 Then you will find the list of **configure options** in the browser.

 In the search bar, type **javascript.enabled**.

 There you will find the option to enable or disable javascript by right-clicking on the value of that option -> **select toggle**.

If javascript.enabled is true; it converts to false upon clicking **toogle**. If javascript is disabled; it gets enabled upon clicking toggle. **19**

**JavaScript in Chrome**

Here are the steps to turn on or turn off JavaScript in Chrome:

 Click the Chrome menu at the top right hand corner of your browser.

 Select **Settings**.

 Click **Show advanced settings** at the end of the page.

 Under the **Privacy** section, click the Content settings button.

 In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

**JavaScript in Opera**

Here are the steps to turn on or turn off JavaScript in Opera:

 Follow **Tools-> Preferences** from the menu.

 Select **Advanced** option from the dialog box.

 Select **Content** from the listed items.

 Select **Enable JavaScript** checkbox.

 Finally click OK and come out.

To disable JavaScript support in Opera, you should not select the **Enable JavaScript checkbox**.

**Warning for Non-JavaScript Browsers**

If you have to do something important using JavaScript, then you can display a warning message to the user using **<noscript>** tags.

You can add a **noscript** block immediately after the script block as follows:

<html>

<body>

<script language="javascript" type="text/javascript">

<!-- **20**

document.write ("Hello World!")

//-->

</script>

<noscript>

Sorry...JavaScript is needed to go ahead.

</noscript>

</body>

</html>

Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from </noscript> will be displayed on the screen. **21**

**4. JAVASCRIPT – PLACEMENT**

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows:

 Script in <head>...</head> section.

 Script in <body>...</body> section.

 Script in <body>...</body> and <head>...</head> sections.

 Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

**JavaScript in <head>...</head> Section**

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows.

<html>

<head>

<script type="text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

Click here for the result

<input type="button" onclick="sayHello()" value="Say Hello" /> **22**

</body>

</html>

This code will produce the following results:

Click here for the result

Say Hello

**JavaScript in <body>...</body> Section**

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

<html>

<head>

</head>

<body>

<script type="text/javascript">

<!--

document.write("Hello World")

//-->

</script>

<p>This is web page body </p>

</body>

</html>

This code will produce the following results:

Hello World **23**

This is web page body

**JavaScript in <body> and <head> Sections**

You can put your JavaScript code in <head> and <body> section altogether as follows.

<html>

<head>

<script type="text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<script type="text/javascript">

<!--

document.write("Hello World")

//-->

</script>

<input type="button" onclick="sayHello()" value="Say Hello" />

</body>

</html>

This code will produce the following result.

HelloWorld

Say Hello **24**

**JavaScript in External File**

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using **script** tag and its **src** attribute.

<html>

<head>

<script type="text/javascript" src="filename.js" ></script>

</head>

<body>

.......

</body>

</html>

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

function sayHello() {

alert("Hello World")

} **25**

**5. JAVASCRIPT – VARIABLES**

**JavaScript Datatypes**

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

 **Numbers**, e.g., 123, 120.50 etc.

 **Strings** of text, e.g. "This text string" etc.

 **Boolean**, e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined**, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

**Note:** Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

**JavaScript Variables**

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

<script type="text/javascript">

<!--

var money;

var name; **26**

//-->

</script>

You can also declare multiple variables with the same **var** keyword as follows:

<script type="text/javascript">

<!--

var money, name;

//-->

</script>

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

<script type="text/javascript">

<!--

var name = "Ali";

var money;

money = 2000.50;

//-->

</script>

**Note:** Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically. **27**

**JavaScript Variable Scope**

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

 **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.

 **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

<script type="text/javascript">

<!--

var myVar = "global"; // Declare a global variable

function checkscope( ) {

var myVar = "local"; // Declare a local variable

document.write(myVar);

}

//-->

</script>

It will produce the following result:

Local

**JavaScript Variable Names**

While naming your variables in JavaScript, keep the following rules in mind. **28**

 You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.

 JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but ***\_*123test** is a valid one.

 JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

**JavaScript Reserved Words**

|  |  |  |  |
| --- | --- | --- | --- |
| A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names. abstract  boolean  break  byte  case  catch  char  class  const  continue  debugger  default  delete  do  double | else  enum  export  extends  false  final  finally  float  for  function  goto  if  implements  import  in | Instanceof  int  interface  long  native  new  null  package  private  protected  public  return  short  static  super | switch  synchronized  this  throw  throws  transient  true  try  typeof  var  void  volatile  while  with |

**8.CREATING A WEB PAGE WITH BACK END IN PHP AND FRONT END IN JAVASCRIPT AND HOSTING IT ON APACHE TOMCAT SERVER**

<?php

echo "function validate\_form() {

<!--

var username = document.getElementById(\\"username\\");

var password = document.getElementById(\\"password\\");

var status = document.getElementById(\\"status\\");

if (username.value.length <= 3) {

status.innerHTML = \\"Username is too short!\\";

username.focus();

return false;

}

else if (password.value.length <= 3) {

status.innerHTML = \\"Password is too short!\\";

password.focus();

return false;

}

else {

return true;

}

//-->

}";

?>

**9.WRITING AND UNDERSTANDING PROGRAMS IN PYTHON**

**FIBONACCI**

first\_no=eval(input("enter first no:"))

second\_no=eval(input("enter second no:"))

limit=int(input("no of fibonacci nos to be print"))

print(first\_no,end=" ")

print(second\_no,end=" ")

for i in range(limit+1):

sum=first\_no+second\_no

first\_no=second\_no

second\_no=sum

print(sum,end=" ")

**10.USE PYTHON LIBRARY LIKE MATH’S STATISTICS TO CREATE PROGRAM**

##Program using Library Maths statics

from statistics import mean

no=[12,23,34,45,56]

mean(no)