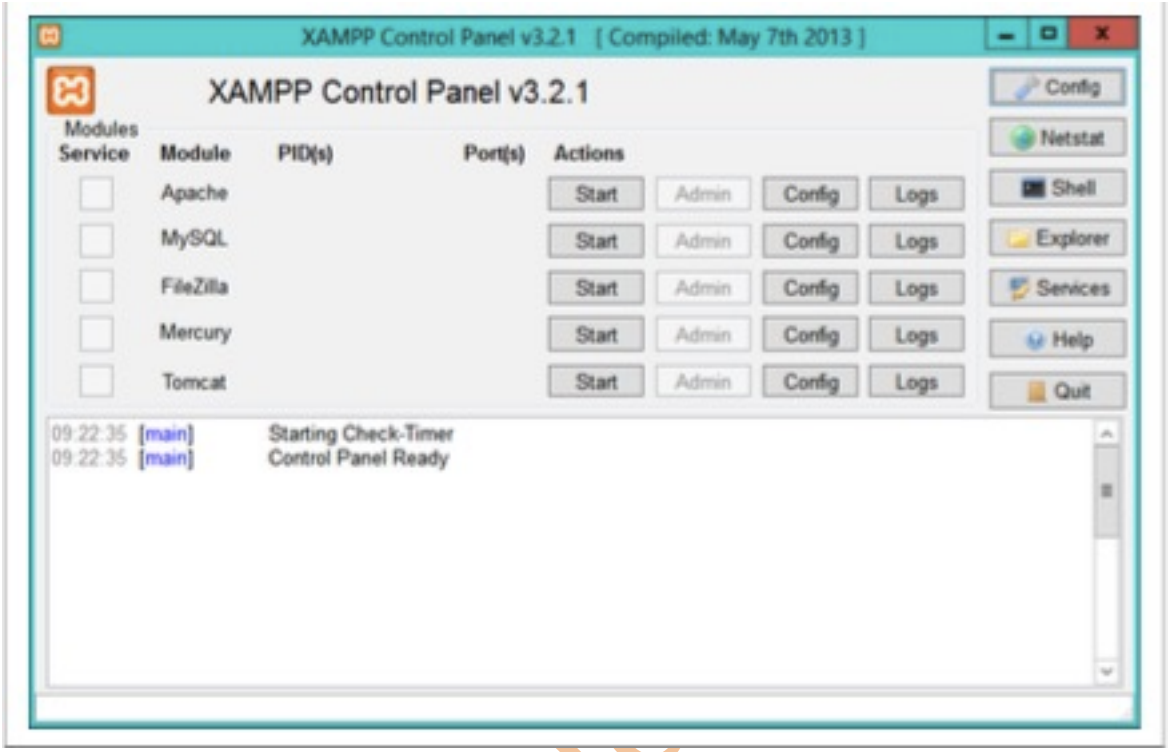
Server Side Programming – PHP

Introduction to PHP

Setting up the XAMPP Development Server

http://apachefriends.org



Using Comments

There are two ways in which you can add comments to your PHP code. The first turns a single line into a comment by preceding it with a pair of forward slashes:

```
// This is a comment
```

This version of the comment feature is a great way to temporarily remove a line of code from a program that is giving you errors.

For example, you could use such a comment to hide a debugging line of code until you need it, like this:

```
// echo "X equals $x";
```

You can also use this type of comment directly after a line of code to describe its action, like this:

```
$x += 10; // Increment $x by 10
```

When you need multiple-line comments, there's a second type of comment, which looks like this:

```
<?php
/* This is a section
   of multiline comments
   which will not be
   interpreted */
?>
```

The $ Symbol

In PHP, you must place a $ in front of all variables. This is required to make the PHP parser faster, as it instantly knows whenever it comes across a variable. Whether your variables are numbers, strings, or arrays, they should all look some-thing like those in following example:

```
<?php
  $mycounter = 1;
  $mystring  = "Hello";
  $myarray   = array("One", "Two", "Three");
?>
```

Variable-naming rules

When creating PHP variables, you must follow these four rules:

1. Variable names must start with a letter of the alphabet or the _ (underscore) character.
2. Variable names can contain only the characters a-z, A-Z, 0-9, and _ (underscore).
3. Variable names may not contain spaces. If a variable must comprise more than one word, it should be separated with the _ (underscore) character (e.g. $user_name).
4. Variable names are case-sensitive. The variable $High_Score is not the same as the variable $high_score.

Refer var.php in notebook

function.php

// Write a function to display a name in Title case.

```
<?php
  echo "<br>";
  echo fix_names1("WILLIAM", "henry", "gatES");

  function fix_names1($n1, $n2, $n3)
  {
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));

    return $n1 . " " . $n2 . " " . $n3;
  }
  echo "<br>";

  $names = fix_names2("sanDeep", "j", "guptA");
  echo $names[0] . " " . $names[1] . " " . $names[2];

  function fix_names2($n1, $n2, $n3)
  {
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));

    return array($n1, $n2, $n3);  // Fn. returns an array
  }
?>
```

The include Statement

Using include, you can tell PHP to fetch a particular file and load all its contents. It's as if you pasted the included file into the current file at the insertion point.

Using include_once

Each time you issue the include directive, it includes the requested file again, even if you've already inserted it. For instance, suppose that library.php contains a lot of useful functions, so you include it in your file, but also include another library that includes library.php. Through nesting, you've inadvertently included library.php twice. This will produce error messages, because you're trying to define the same constant or function multiple times. So you should use include_once instead.

Then, whenever another include or include_once is encountered, if it has already been executed, it will be completely ignored. To determine whether the file has already been executed, the absolute file path is matched after all relative paths are resolved and the file is found in your include path.
In general, it's probably best to stick with include_once and ignore the basic include statement. That way, you will never have the problem of files being included multiple times.

Using require and require_once

A potential problem with include and include_once is that PHP will only attempt to include the requested file. Program execution continues even if the file is not found.
When it is absolutely essential to include a file, require it. It is recommended that you stick to require_once whenever you need to require a file.

include_require.php

```
// Demonstrate usage of include

<?php
  echo "<br>";
  include "function.php";
  echo "<br>";
  echo fix_names1("pesHwa", "bajirao", "bhosAle");
  echo "<br>";
  $names = fix_names2("meiN", "hoon", "dagadChand");
  echo $names[0] . " " . $names[1] . " " . $names[2];
  echo "<br>";
?>

/* Instead of include "function.php", we could have also written
include_once "function.php";
require "function.php";
include_once "function.php";
*/
```

Arrays

Refer array.php

Classes and Objects

class.php

```
/*Demonstarate classes and objects */

<?php

class Box
{
  private  $width;
  var  $height;          // var means public
  public  $depth;

  function  __construct($w, $h, $d)
  {
    $this->width = $w;
    $this->height = $h;
    $this->depth  = $d;
  }

  function volume()
  {
    $v = $this->width * $this->height * $this->depth ;
    return $v;
  }
}

$b1 = new Box(2,3,4);
$b2 = new Box(5,6,2);

print_r($b1);
echo "<br>";

print_r($b2);
echo "<br>";

echo  $b1->volume();
echo "<br>";
echo  $b2->volume();
echo "<br>";
//echo  $b2->width;
echo  $b1->height;
echo "<br>";
echo  $b1->depth;
?>
```

cloning.php

```
/*Demonstarate object cloning*/

<?php
  $object1 = new User();
  $object1->name = "Alice";
  $object2 = $object1;
  $object2->name = "Amy";

  echo "object1 name =  $object1->name <br>";
  echo "object2 name =  $object2->name <br>";

  class User
  {
    public  $name;
  }

  $object3  = new User();
  $object3->name = "Jack";
  $object4  = clone $object3;
  $object4->name = "Jill";

  echo "object3 name =  $object3->name <br>";
  echo "object4 name =  $object4->name <br>";
?>
```

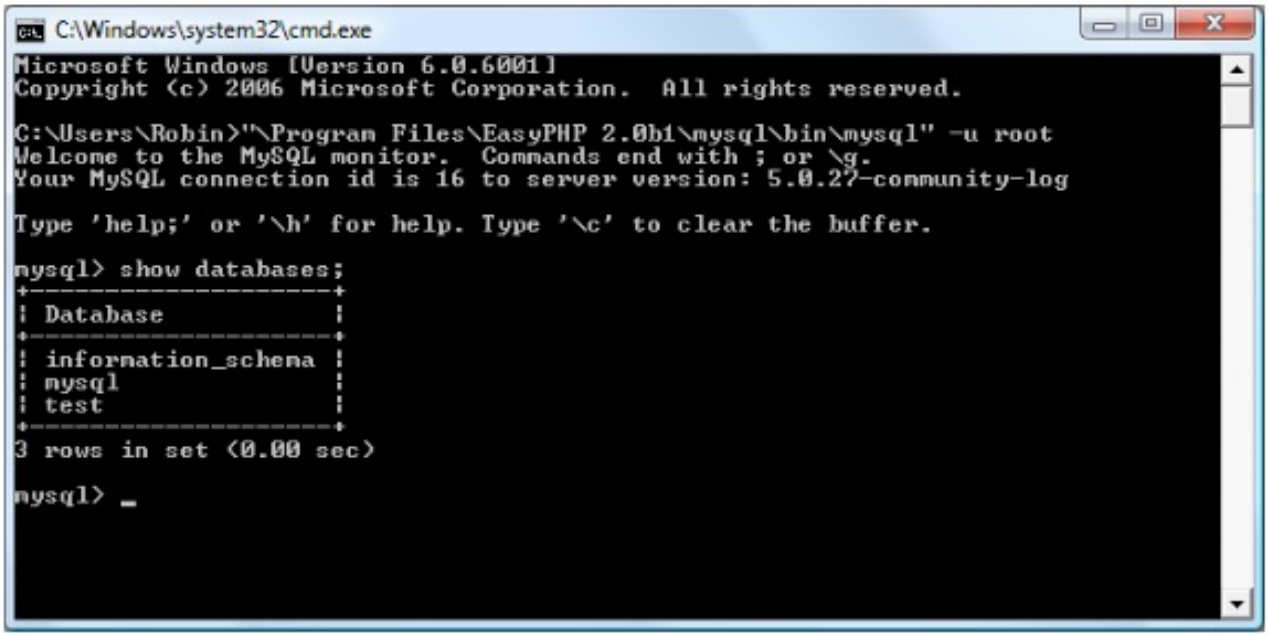
Accessing MySQL using Command Line Interface

Windows users:

To enter MySQL's command-line interface, select Start→Run, enter CMD into the Run box, and press Return. This will call up a Windows command prompt. From there, enter

C:\xampp\mysql\bin\mysql -u root

This command tells MySQL to log you in as user root, without a password. You will now be logged into MySQL and can start entering commands. So, to be sure everything is working as it should be, enter the following: SHOW databases;



Creating a database

Example: CREATE DATABASE publications;
A successful command will return a message that doesn't mean much—
Query OK, 1 row affected (0.00 sec).
Now that you've created the database, you want to work with it, so issue the following:
USE publications;
You should now see the message Database changed.

Creating users

Now that you've seen how easy it is to use MySQL, and created your first database, it's time to look at how you create users, as you probably won't want to grant your PHP scripts root access to MySQL; it could cause a real headache should you get hacked.

So let's create a user who can access just the new publications database by entering something like this:

GRANT ALL ON publications.* TO '**sandeep**'@'localhost' IDENTIFIED BY '**studycircle**';

Change red colour as per your requirement. Here sandeep is username and studycircle is password.

Accessing MySQL using PHP

Creating a Login File

Most websites developed with PHP contain multiple program files that will require access to MySQL and will thus need the login and password details. Therefore, it's sensible to create a single file to store these and then include that file wherever it's needed.

login.php

```
<?php
$hn = 'localhost';
$un = 'sandeep';
$pw = 'studycircle';
$db = 'employeedb';
?>
```

Refer query.php from notebook