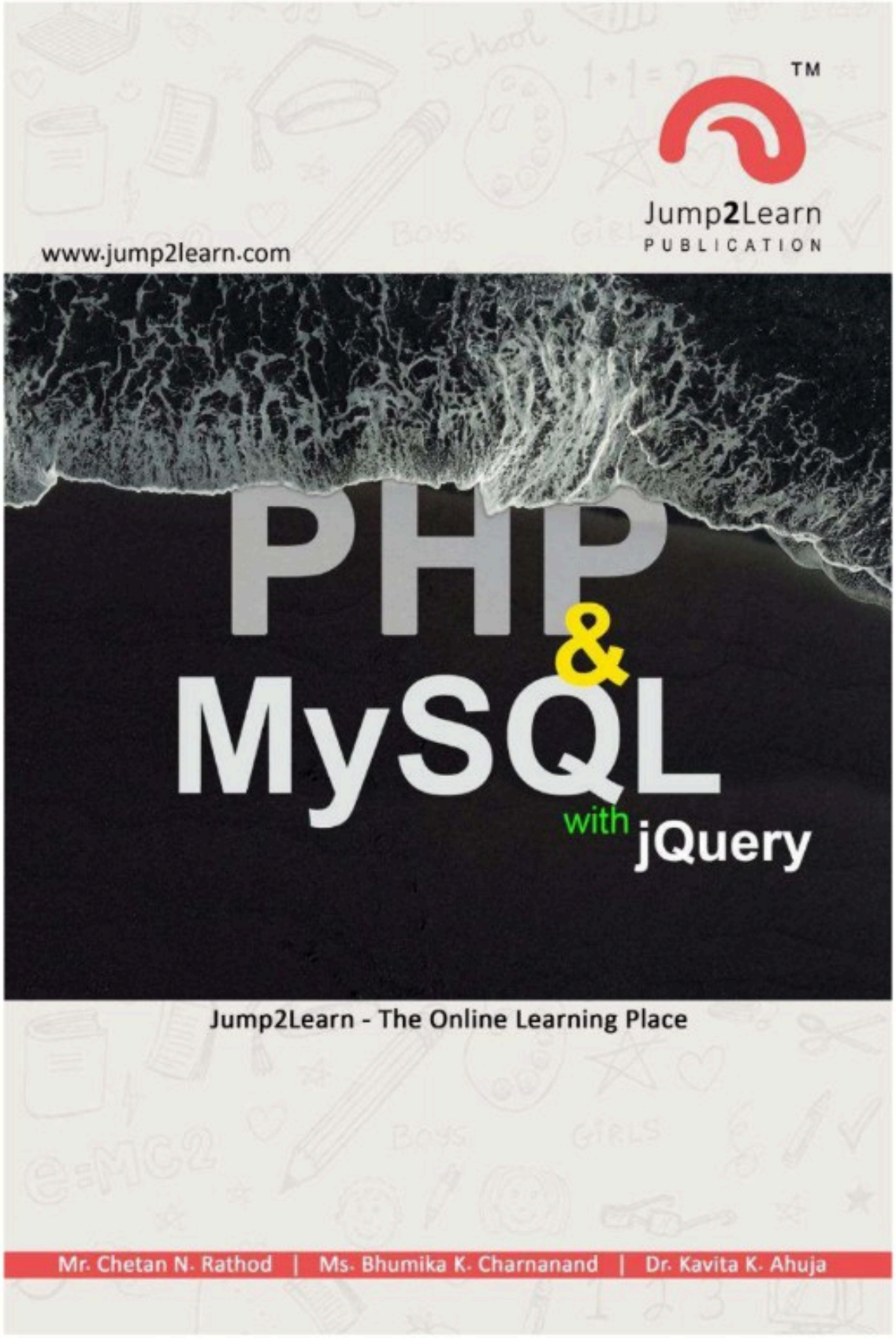
Jump2Learn
PUBLICATION
www.jump2learn.com
PHP
&
MySQL
with jQuery
Jump2Learn - The Online Learning Place
Mr. Chetan N. Rathod | Ms. Bhumika K. Charnanand | Dr. Kavita K. Ahuja

Unit - 3

Session, Cookies And Upload Files

Concept of Session
Starting session
Storing Session Variable
Modifying session variables
Unregistering and deleting session variable
Concept of Cookies and Querystring
Upload file form
Uploading scripts and restrictions on upload
Saving uploaded file

SESSION

A PHP session variable is used to store information about settings for a user session, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

PHP Session Variables

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc.). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Starting session

Before you can store user information in your PHP session, you must first start up the session.

Note: The session_start() function must appear BEFORE the <html> tag:


```
<?php session_start(); ?>
<html>
<body>

</body>
</html>
```

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

STORING A SESSION VARIABLE

$_SESSION

The correct way to store and retrieve session variables is to use the PHP $_SESSION variable:

Syntax:

```
$_SESSION['variable name']=value;
```

Example:

```
<?php
    session_start();
    // store session data
    $_SESSION['views']=1;
?>
<html>
<body>
<?php
    //retrieve session data
    echo "Pageviews=". $_SESSION['views'];
?>
</body>
</html>
```

Output:

```
Pageviews=1
```

In the example below, we create a simple page-views counter.

MODIFYING SESSION VARIABLES

We can change the value of session variable by increment it (or by setting another value)

Example:

```
$_SESSION['views']=$_SESSION['views']+1;
```

isset()

The isset() function checks if the "views" variable has already been set. If "views" has been set, we can increment our counter. If "views" doesn't exist, we create a "views" variable, and set it to 1:

```
<?php
	session_start();
		if(isset($_SESSION['views']))
			$_SESSION['views']=$_SESSION['views']+1;
		else
			$_SESSION['views']=1;
	echo "Views=". $_SESSION['views'];
?>
```

UNREGISTERING AND DELETING SESSION VARIABLE

If you wish to delete some session data, you can use the unset() or the session_destroy() function.

unset()

The unset() function is used to free the specified session variable:

```
<?php
	unset($_SESSION['views']);
?>
```


session_destroy()

You can also completely destroy the session by calling the session_destroy() function:

```
<?php
	session_destroy();
?>
```

Note: session_destroy() will reset your session and you will lose all your stored session data.

COOKIES

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

CREATE A COOKIE

The setcookie() function is used to set a cookie.

Note: The setcookie() function must appear BEFORE the <html> tag.

Syntax:

setcookie(name, value, expire, path, domain);

Example 1:

In the example below, we will create a cookie named "user" and assign the value "Chetan Rathod" to it. We also specify that the cookie should expire after one hour:

```
<?php
    setcookie("user", "Chetan Rathod", time()+3600);
?>
<html>
...
```


Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Example 2:

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.

```
<?php
    $expire=time()+60*60*24*30;
    setcookie("user", "Chetan Rathod", $expire);
?>
```

```
<html>
.....
```

In the example above the expiration time is set to a month (60 sec * 60 min * 24 hours * 30 days).

RETRIEVE A COOKIE VALUE

$_COOKIE

The PHP $_COOKIE variable is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "user" and display it on a page:

```
<?php
        // Print a cookie
        echo $_COOKIE["user"];

        // A way to view all cookies
        print_r($_COOKIE);
?>
```

In the following example we use the isset() function to find out if a cookie has been set:

```
<html>
<body>
<?php
        if (isset($_COOKIE["user"]))
                echo "Welcome " . $_COOKIE["user"] . "!<br />";
        else
                echo "Welcome guest!<br />";
?>
</body>
</html>
```

DELETE A COOKIE

When deleting a cookie you should assure that the expiration date is in the past.
Delete example:

```
<?php
        // set the expiration date to one hour ago
        setcookie("user", "", time()-3600);
?>
```

Note: If your application deals with browsers that do not support cookies, you will have to use other methods to pass information from one page to another in your application

UPLOAD FILE FORM

With PHP, it is possible to upload files to the server.

CREATE AN UPLOAD-FILE FORM

To allow users to upload files from a form can be very useful.

Look at the following HTML form for uploading files:

```
<html>
<body>
    <form action="upload_file.php" method="post" enctype="multipart/form-data">
        <label for="file">Filename:</label>
        <input type="file" name="file" id="file" />
        <br />
        <input type="submit" name="submit" value="Submit" />
    </form>
</body>
</html>
```


NOTICE THE FOLLOWING ABOUT THE HTML FORM ABOVE:

- The **enctype** attribute of the <form> tag specifies which content-type to use when submitting the form. "multipart/form-data" is used when a form requires binary data, like the contents of a file, to be uploaded

- The **type="file"** attribute of the <input> tag specifies that the input should be processed as a file. For example, when viewed in a browser, there will be a browse-button next to the input field

Note: Allowing users to upload files is a big security risk. Only permit trusted users to perform file uploads.

CREATE THE UPLOAD SCRIPT

The "upload_file.php" file contains the code for uploading a file:

```
<?php
	if ($_FILES["file"]["error"] > 0)
	{
		echo "Error: " . $_FILES["file"]["error"] . "<br />";
	}
	else
	{
		echo "Upload: " . $_FILES["file"]["name"] . "<br />";
		echo "Type: " . $_FILES["file"]["type"] . "<br />";
		echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
		echo "Stored in: " . $_FILES["file"]["tmp_name"];
	}
?>
```

By using the global PHP $_FILES array you can upload files from a client computer to the remote server.

The first parameter is the form's input name and the second index can be either "name", "type", "size", "tmp_name" or "error". Like this:

- $_FILES["file"]["name"] - the name of the uploaded file
- $_FILES["file"]["type"] - the type of the uploaded file
- $_FILES["file"]["size"] - the size in bytes of the uploaded file
- $_FILES["file"]["tmp_name"] - the name of the temporary copy of the file stored on the server
- $_FILES["file"]["error"] - the error code resulting from the file upload

This is a very simple way of uploading files. For security reasons, you should add restrictions on what the user is allowed to upload.

RESTRICTIONS ON UPLOAD

In this script we add some restrictions to the file upload. The user may only upload .gif or .jpeg files and the file size must be under 20 kb:

```
<?php
    if ((($_FILES["file"]["type"] == "image/gif") || ($_FILES["file"]["type"] ==
    "image/jpeg") || ($_FILES["file"]["type"] == "image/pjpeg")) &&
    ($_FILES["file"]["size"] < 20000))
    {
        if ($_FILES["file"]["error"] > 0)
        {
            echo "Error: " . $_FILES["file"]["error"] . "<br />";
        }
        else
        {
            echo "Upload: " . $_FILES["file"]["name"] . "<br />";
            echo "Type: " . $_FILES["file"]["type"] . "<br />";
            echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
            echo "Stored in: " . $_FILES["file"]["tmp_name"];
        }
    }
    else
    {
        echo "Invalid file";
    }
?>
```

Note: For IE to recognize jpg files the type must be pjpeg, for FireFox it must be jpeg.

SAVING UPLOADED FILE

The examples above create a temporary copy of the uploaded files in the PHP temp folder on the server.

The temporary copied files disappears when the script ends. To store the uploaded file we need to copy it to a different location:

```
<?php
```

```
if ((($_FILES["file"]["type"] == "image/gif") || ($_FILES["file"]["type"] ==
"image/jpeg") || ($_FILES["file"]["type"] == "image/pjpeg")) &&
($_FILES["file"]["size"] < 20000))
{
        if ($_FILES["file"]["error"] > 0)
        {
                echo "Return Code: " . $_FILES["file"]["error"] . "<br />";
        }
        else
        {
                echo "Upload: " . $_FILES["file"]["name"] . "<br />";
                echo "Type: " . $_FILES["file"]["type"] . "<br />";
                echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
                echo "Temp file: " . $_FILES["file"]["tmp_name"] . "<br />";
                        if (file_exists("upload/" . $_FILES["file"]["name"]))
                        {
                                echo $_FILES["file"]["name"] . " already exists. ";
                        }
                        else
                        {
                                move_uploaded_file($_FILES["file"]["tmp_name"
                                ], "upload/" . $_FILES["file"]["name"]);
                                echo "Stored in: " . "upload/" .
                                $_FILES["file"]["name"];
                        }
        }
}
else
{
        echo "Invalid file";
}
?>
```

The script above checks if the file already exists, if it does not, it copies the file to the specified folder.

Note: This example saves the file to a new folder called "upload"