Lecture-21-22

Functions in PHP: Defining a Function , Calling A function, Variable Scope(Global variable& Static Variables)

Function

- A function is a piece of code in a larger program. The function performs a specific task. The advantages of using functions are:
- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

There are four basic types of functions.

- **user-defined functions**
- **Function arguments**
- **Returning values**
- **Variable functions**
- **Internal (built-in) functions**
- **Anonymous functions**

PHP User Defined Functions

- Besides the built-in PHP functions, we can create our own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.

- Any valid PHP code may appear inside a function, even other functions and class definitions.
- Function names follow the same rules as other labels in PHP. A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

Defining functions

- A function is created with the function keyword.
- A function may be defined using syntax such as the following:

```
<?php
   function functionName
   ($arg_1, $arg_2, /* ..., */ $arg_n)
   {
      echo "Example function.\n";
      return $retval;
   }
   ?>
```

Conditional functions

- Functions need not be defined before they are referenced, *except* when a function is conditionally defined.
- When a function is defined in a conditional manner. Its definition must be processed *prior* to being called.

Example: Conditional functions

```
<?php

$makefunction = true;

/* We can't call fun() from here
   since it doesn't exist yet,
   but we can call bar() */
bar();
//fun();
if ($makefunction) {
  function fun()
  {
    echo "I don't exist until program execution reaches me <br>";
  }
```

```
/* Now we can safely call fun()
   since $makefunction evaluated to true */

if ($makefunction)
fun();

function bar()
{
  echo "I exist immediately upon program start.<br>";
}

?>
```

Example: Functions within functions

```
<?php
function abc()
{
echo "hello <br>";
  function xyz()
  {
    echo "I don't exist until abc() is called.\n";
  }
}

/* We can't call xyz() yet
   since it doesn't exist. */

abc();

/* Now we can call xyz(),
   abc()'s processing has
   made it accessible. */

xyz();

?>
```

Note:

- All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa.
- PHP does not support function overloading, nor is it possible to undefine or redefine previously-declared functions.

- Function names are case-insensitive, though it is usually good form to call functions as they appear in their declaration.
- Both variable number of arguments and default arguments are supported in functions.
- It is possible to call recursive functions in PHP. However avoid recursive function/method calls with over 100-200 recursion levels as it can smash the stack and cause a termination of the current script.

Example :Recursive functions

```
  <?php
function recursion($a)
{
    if ($a < 20) {
        echo "$a\n";
        recursion($a + 1);
    }
}
recursion(1);
?>
```

Function arguments

- Information may be passed to functions via the argument list, which is a comma-delimited list of expressions.
- The arguments are evaluated from left to right.
- PHP supports passing arguments by value (the default), passing by reference, and default argument values.
- Variable-length argument lists are also supported.

Example: function with one argument

```
<?php
function familyName($fname)
{
echo "$fname Sharma.<br>";
}

familyName(“Aman");
familyName(“Ajay");
familyName(“Ankit");
familyName(“Anil");
familyName(“Amar");
?>
```

Example :function with two arguments

```
<?php
function familyName($fname,$year)
{
echo "$fname Sharma. Born in $year <br>";
}

familyName(“Aman","1975");
familyName(“Amar","1978");
familyName(“Ajay","1983");
?>
```

PHP Default Argument Value

```
<?php
   function  setHeight($minheight=50)
   {
   echo "The height is : $minheight <br>";
   }
   setHeight(350);
   setHeight(); // will use the default value of 50
   setHeight(135);
   setHeight(80);
   ?>
```

Example : Passing arrays to functions

```
<?php
   function takes_array($input)
   {
 echo "$input[0] + $input[1] = ", $input[0]+$input[1];
   }
$xyz=array[“a”,”b”]
takes_array($xyz);
   ?>
```

function arguments are passed by value or Call by Value

```
<?php
$a=10;
$b=20;
echo "before swaping value is : $a  $b <br>";
function swap($i,$j)
{
$t=$i;
$i=$j;
$j=$t;
echo "after swaping $i $j";
}
swap($a, $b)
?>
```

Making arguments be passed by reference

- By default, function arguments are passed by value (so that if the value of the argument within the function is changed, it does not get changed outside of the function).
- To allow a function to modify its arguments, they must be passed by reference.
- To have an argument to a function always passed by reference, prepend an ampersand (&) to the argument name in the function definition:

Call By Reference

```
<?php
$a=10;
$b=20;
echo "before swaping value is : $a  $b <br>";
function swap(&$i ,& $j)
{
$t=$i;
$i=$j;
$j=$t;
echo "after swaping $i $j";
}
swap($a, $b)
?>
```

Example : Passing function parameters by reference

```
<?php
   function add_some_extra(&$string)
   {
      $string .= 'and something extra.';
   }
   $str = 'This is a string, ';
   add_some_extra($str);
   echo $str;    // outputs 'This is a string, and something e
   xtra.'
   ?>
```

Example: returning value from call by references

```
<?php
$a1="William";
$a2="henry";
$a3="gates";
echo $a1." ".$a2." ".$a3 ."<br/>" ;
fix_name($a1,$a2,$a3);
echo $a1." ".$a2." ".$a3."<br>";

function fix_name(&$n1,&$n2,&$n3)
{
$n1=strtoupper($n1);
$n2=strtoupper($n2);
$n3=strtoupper($n3);

}
?>
```

PHP also allows the use of arrays and the special type NULL as default values, for example:

```
 <?php
function makecoffee($types = array("cappuccino"), $coffeeMaker =
   NULL)
{
   $device = is_null($coffeeMaker) ? "hands" : $coffeeMaker;
   echo"<br>";
  return "Making a cup of ".join(", ", $types)." with $device.<br>";
}
   echo makecoffee();

echo makecoffee(array("cappuccino", "lavazza"), "teapot");
?>
```

Note:

- The default value must be a constant expression, not (for example) a variable, a class member or a function call.
- when using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected.

Example: Incorrect usage of default function arguments

```
<?php
   function makeyogurt($type = "acidophilus", $flavour)
   {
      return "Making a bowl of $type $flavour.\n";
   }

   echo makeyogurt("raspberry");   // won't work as expected
   ?>
```

- **output:**

```
 Warning: Missing argument 2 for makeyogurt(), called in
C:\wamp\www\suman\fun_Incorrect_default_null.php on line 8 and
defined in C:\wamp\www\suman\fun_Incorrect_default_null.php
on line 3

Notice: Undefined variable: flavour in
C:\wamp\www\suman\fun_Incorrect_default_null.php on line 5
Making a bowl of berry
```

Example Correct usage of default function arguments

```
<?php
function makeyogurt($flavour, $type = "acid")
{
    return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt("berry");   // works as expected
?>
```

output:

- Making a bowl of acid berry.

Note: **In PHP 5, arguments that are passed by reference may have a default value.**

- **Variable-length argument lists**
- PHP has support for variable-length argument lists in user-defined functions. This is really quite easy, using the func_num_args(), func_get_arg(), and func_get_args() functions.
- No special syntax is required, and argument lists may still be explicitly provided with function definitions and will behave as normal.

3. Returning values

- Values are returned by using the optional return statement. Any type may be returned, including arrays and objects. This causes the function to end its execution immediately and pass control back to the line from which it was called.
- **Note**: If the return is omitted the value **NULL** will be returned.

Example :Use of return

```
<?php
   function square($num)
   {
       return $num * $num;
   }
   echo square(4);   // outputs '16'.
   ?>
```

A function can not return multiple values, but similar results can be obtained by returning an array.

- **Example: Returning an array to get multiple values**

```
<?php
function small_numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
?>
```

- To return a reference from a function, use the reference operator & in both the function declaration and when assigning the returned value to a variable:
- **Example :Returning a reference from a function**

```
<?php
function &returns_reference()
{
    return $someref;
}
$newref =& returns_reference();
?>
```

4: Variable functions

- PHP supports the concept of variable functions.
- This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.
- Among other things, this can be used to implement callbacks, function tables, and so forth.
- Variable functions won't work with language constructs such as echo, print, unset(), isset(), empty(), include, require and the like. Utilize wrapper functions to make use of any of these constructs as variable functions.

Example :Variable function example

```
 <?php
function abc() {
   echo "In abc()<br />\n";
}

function xyz($arg = '')
{
   echo "In xyz(); argument was '$arg'.<br />\n";
}

// This is a wrapper function around echo
function echoit($string)
{
   echo $string;
}

$func = 'abc';
$func();

$func = 'xyz';
$func('test');
$func = 'echoit';
$func('test');
?>
```

Object methods can also be called with the variable functions syntax.

Example : Variable method example

```
<?php
class Foo
{
    function Variable()
    {
        $name = 'Bar';
        $this->$name(); // This calls the Bar() method
    }

    function Bar()
    {
        echo "This is Bar";
    }
}

$foo = new Foo();
$funcname = "Variable";
$foo->$funcname();  // This calls $foo->Variable()

?>
```

When calling static methods, the function call is stronger than the static property operator:

- **Example: Variable method example with static properties**

```
<?php
   class Foo
   {
      static $variable = 'static property';
      static function Variable()
      {
         echo 'Method Variable called';
      }
   }

   echo Foo::$variable; // This prints 'static property'. It does need a $vari
   able in this scope.
   $variable = "Variable";
   Foo::$variable();  // This calls $foo-
   >Variable() reading $variable in this scope.

   ?>
```

Internal (built-in) functions

- PHP comes standard with many functions and constructs. There are also functions that require specific PHP extensions compiled in, otherwise fatal "undefined function" errors will appear.
- For example:

1. to use image functions such as imagecreatetruecolor(), PHP must be compiled to use mysql_connect(), PHP must be compiled with MySQL support.
2. There are many core functions that are included in every version of PHP, such as the string and variable functions.
3. A call to phpinfo() or get_loaded_extensions() will show which extensions are loaded into PHP. Also note that many extensions are enabled by default and that the PHP manual is split up by extension.

- For example, str_replace() will return the modified string while usort() works on the actual passed in variable itself. Each manual page also has specific information for each function like information on function parameters, behavior changes, return values for both success and failure, and availability information.
- **Note**: If the parameters given to a function are not what it expects, such as passing an array where a string is expected, the return value of the function is undefined. In this case it will likely return **NULL** but this is just a convention, and cannot be relied upon.

Anonymous functions

- Anonymous functions, also known as *closures*, allow the creation of functions which have no specified name. They are most useful as the value of **callback** parameters, but they have many other uses.
- **Example :Anonymous function example**

-
  ```
  <?php
  echo preg_replace_callback('~-([a-
  z])~', function ($match) {
      return strtoupper($match[1]);
  }, 'hello-world');      // outputs helloWorld
  ?>
  ```

Lecture-25
Include , Require, Date ,time and printf function

Ms. Suman Singh
Asst. Professor(IT)

Include and require statement

- In PHP programming we start building a library of functions that we think we will need again.
- We can use inbuilt library function
- There is no need to copy and paste these functions into our code .We can save them and use command to pull.

There are two types of command

1. include statement
2. require statement

Both include and require statement are identical , except upon failure

- require will produce a fatal error (E_COMPILE_ERROR) and stop the script
- include will only produce a warning (E_WARNING) and the script will continue

Include() function

- Using include we can fetch a particular file and load all its contents
- Syntax:

```
Include "filename”;
```

Example:

```
<html>
<body>
<?php include 'header.php'; ?>
<h1>Welcome to my home page!</h1>
<p>Some text.</p>
</body>
</html>
```

Example: we have an include file with some variables defined ("vars.php"):

```
<?php
$color='red';
$car='BMW';
?>
```

Then the variables can be used in the calling file

```
<html>
   <body>
<h1>Welcome to my home page.</h1>
   <?php include 'vars.php';
   echo "I have a $color $car";
   ?>
</body>
   </html>
```

require() Function

- The require() function takes all the text in a specified file and copies it into the file that uses the include function.
- If there is any problem in loading a file then the **require()** function generates a fatal error and halt the execution of the script.
- there is no difference in require() and include() except they handle error conditions.
- It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

Syntax:

require '*filename*';

- Example with require() function following two examples where file does not exist then we will get different results.

Example:

-
  ```
  <html>
  <body>
  <?php include("xxmenu.php"); ?>
  <p>This is an example to show how to include wrong PHP file!</p>
  </body>
  </html>
  ```
- This will produce following result

  ```
  This is an example to show how to include wrong PHP file!
  ```

Example:

```
<html>
<body>
<?php require("xxmenu.php"); ?>
<p>This is an example to show how to include wrong
PHP file!</p>
</body>
</html>
```

This time file execution halts and nothing is displayed.

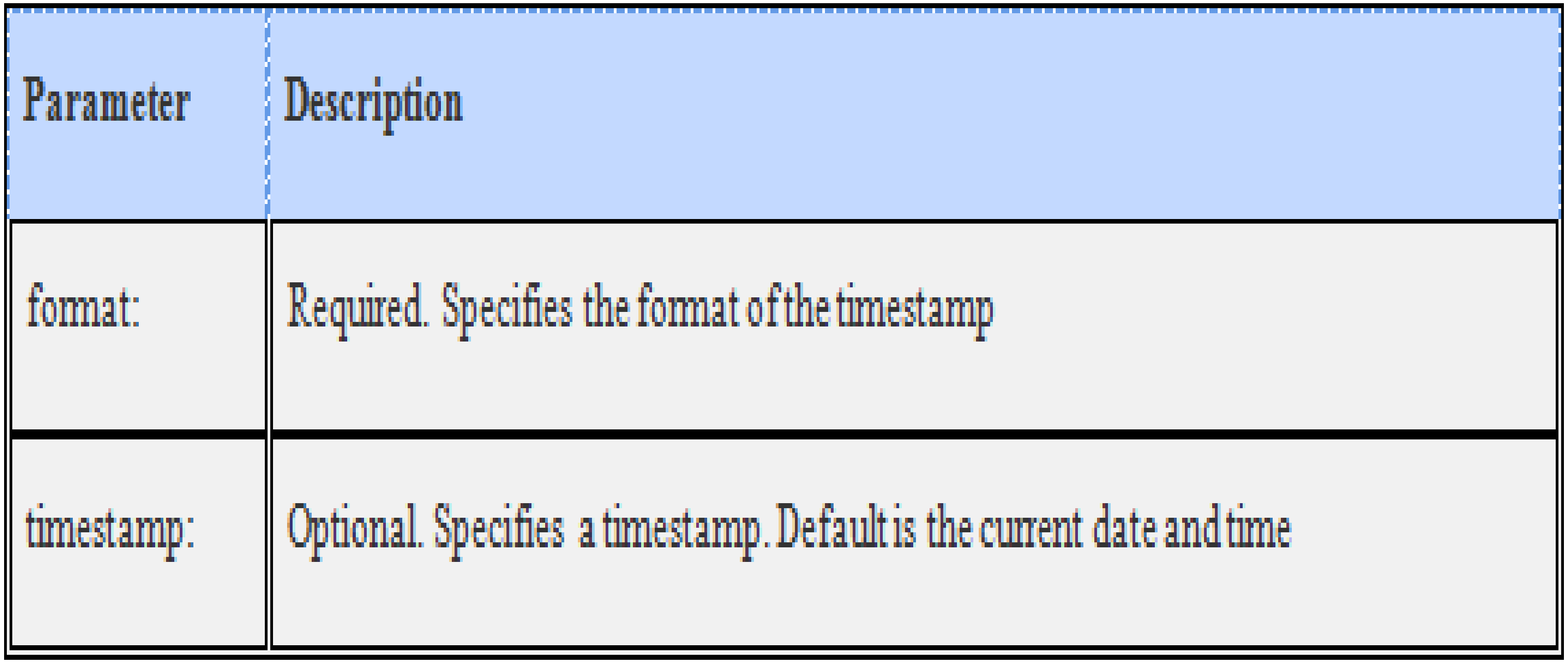
NOTE: You may get plain warning messages or fatal error messages or nothing at all. This depends on our PHP Server configuration.

The PHP Date() Function

- The PHP date() function formats a timestamp to a more readable date and time.
- A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.
- **Syntax**

  date (*format*, *timestamp*)

The PHP Date() Function

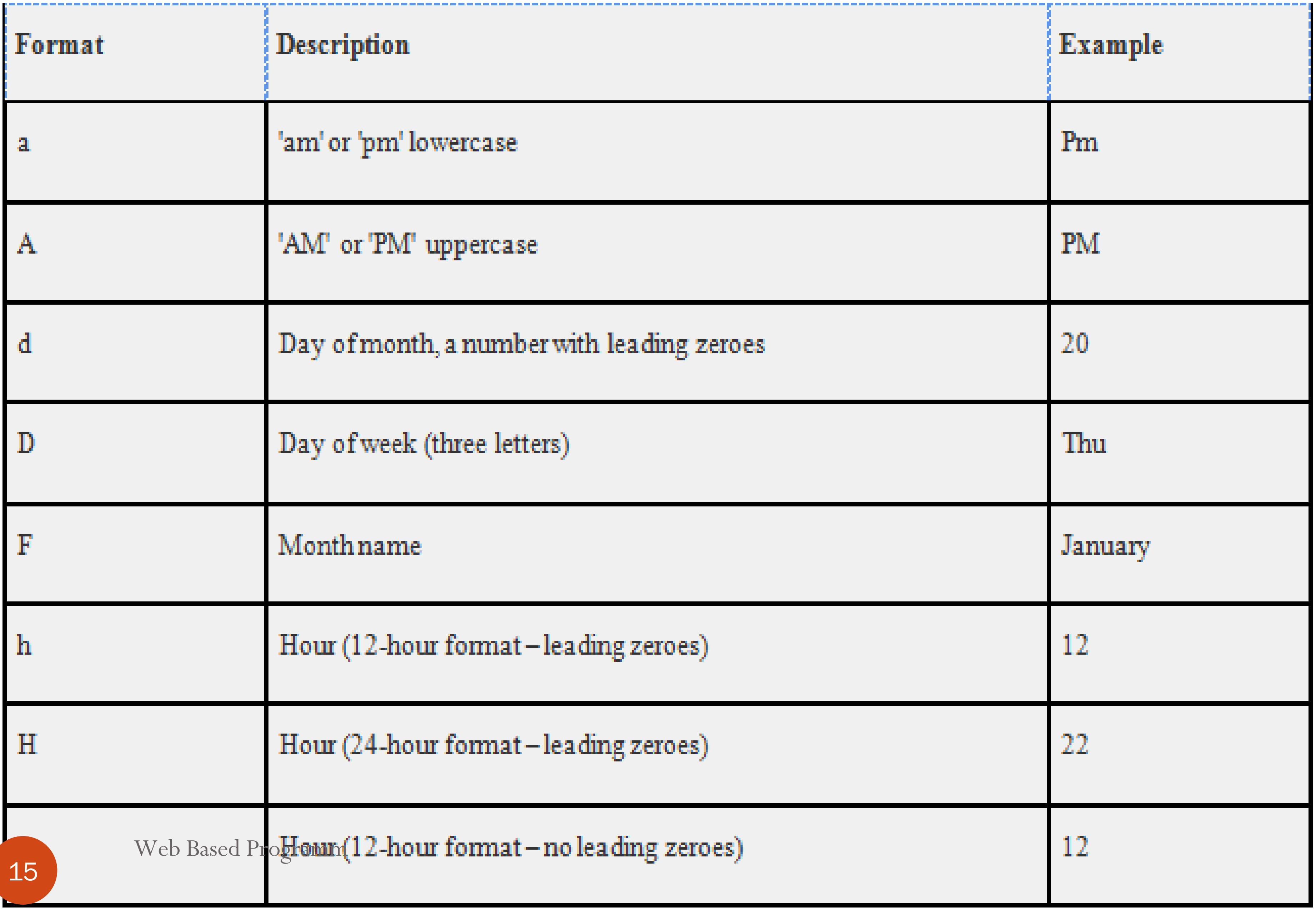


Parameter	Description
format:	Required. Specifies the format of the timestamp
timestamp:	Optional. Specifies a timestamp. Default is the current date and time

Note:

- The date() optionally accepts a time stamp if omitted then current date and time will be used.
- Any other data we include in the format string passed to date() will be included in the return value

Following table lists the codes that a format string can contain:



Format	Description	Example
a	'am' or 'pm' lowercase	Pm
A	'AM' or 'PM' uppercase	PM
d	Day of month, a number with leading zeroes	20
D	Day of week (three letters)	Thu
F	Month name	January
h	Hour (12-hour format – leading zeroes)	12
H	Hour (24-hour format – leading zeroes)	22
	Hour (12-hour format – no leading zeroes)	12

G	Hour (24-hour format – no leading zeroes)	22
i	Minutes (0 – 59)	23
j	Day of the month (no leading zeroes	20
l (Lower 'L')	Day of the week	Thursday
L	Leap year ('1' for yes, '0' for no)	1
m	Month of year (number – leading zeroes)	1
M	Month of year (three letters)	Jan
r	The RFC 2822 formatted date	Thu, 21 Dec 2000 16:01:07 +0200
n	Month of year (number – no leading zeroes)	2
s	Seconds of hour	20
U	Time stamp	948372444
y	Year (two digits)	06
Y	Year (four digits)	2006
z	Day of year (0 – 365)	206
[illegible]	Offset in seconds from GMT	+5

Example:

```
<?php
  print date("m/d/y G.i:s <br>", time());
print "Today is ";
print date("j of F Y, \a\\t g.i a", time());
?>
```

result:

- 01/20/00 13.27:55
- Today is 20 of January 2000, at 1.27 pm

PHP Date() – Adding a Timestamp

- The optional *timestamp* parameter in the date() function specifies a timestamp. If we do not specify a timestamp, the current date and time will be used.
- The mktime() function returns the Unix timestamp for a date.
- The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Syntax for mktime()

mktime(hour, minute, second, month, day , year, is_dst)

Example:

```
<?php
    $tomorrow =
    mktime(0,0,0,date("m"),date("d")+1,date("Y"));
    echo "Tomorrow is ".date("Y/m/d", $tomorrow);
    ?>
```

- The output of the code above could be something like this:

```
Tomorrow is 2009/05/12
```

PHP Time() Function

- PHP's **time()** function gives us all the information that we need about the current date and time.
- It requires no arguments but returns an integer.
- The integer returned by time() represents the number of seconds elapsed since midnight GMT on January 1, 1970. This moment is known as the UNIX epoch, and the number of seconds that have elapsed since then is referred to as a time stamp.

Example:

```
<?php
print time();
?>
```

Output:

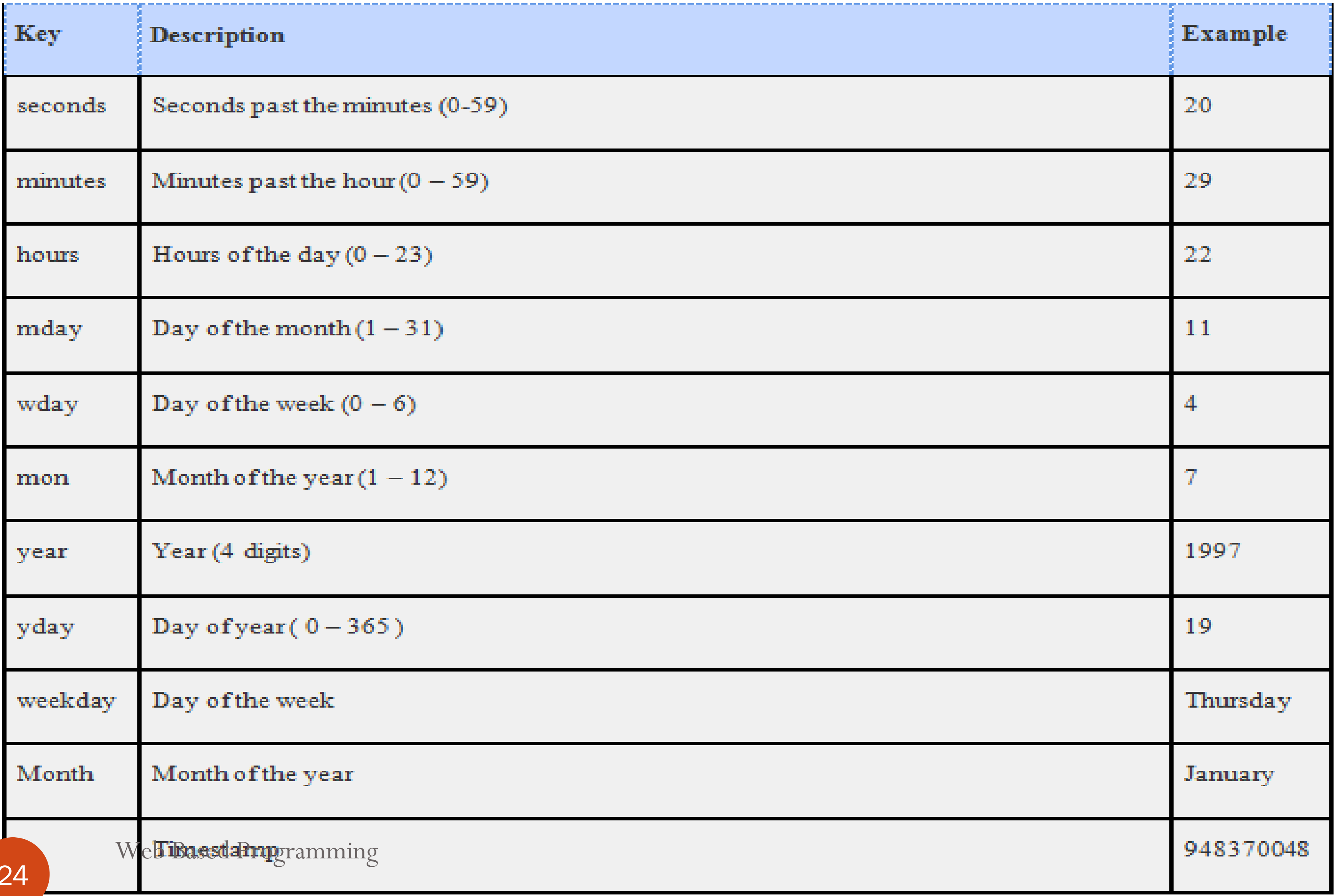
```
948316201
```

Note: This is something difficult to understand. But PHP offers excellent tools to convert a time stamp into a form that humans are comfortable with.

Converting a Time Stamp with getdate():

- The function **getdate()** optionally accepts a time stamp and returns an associative array containing information about the date.
- If we omit the time stamp, it works with the current time stamp as returned by time().

Following table lists the elements contained in the array returned by getdate().



Key	Description	Example
seconds	Seconds past the minutes (0-59)	20
minutes	Minutes past the hour (0 – 59)	29
hours	Hours of the day (0 – 23)	22
mday	Day of the month (1 – 31)	11
wday	Day of the week (0 – 6)	4
mon	Month of the year (1 – 12)	7
year	Year (4 digits)	1997
yday	Day of year (0 – 365)	19
weekday	Day of the week	Thursday
Month	Month of the year	January
	Timestamp	948370048

Example:

```
<?php
$date_array = getdate();
foreach ( $date_array as $key => $val )
{
print "$key = $val<br />";
}
$formated_date  = "Today's date: ";
$formated_date .= $date_array[ mday ] . "/";
$formated_date .= $date_array[ mon] . "/";
$formated_date .= $date_array[year];
print $formated_date;
?>
```

It will produce following result:

```
seconds = 27
minutes = 25
hours = 11
mday = 12
wday = 6
mon = 5
year = 2007
yday = 131
weekday = Saturday
month = May
0 = 1178994327
Today's date: 12/5/2007
```

Printf function

- Print and echo function simply output text to the browser.
- Printf function controls the format of the output by using special characters in a string.
- For each formatting character printf expects to pass an argument that it will display format.
- Example:

```
 printf(“there are %d items in your basket”, 3);
```

List of printf conversion specifier.

Specifier	Conversion action on argument argument
%	Display a % character(no argument is required)
b	Display argument as a binary integer
c	Display the ASCII character for the argument
d	Display arguments as a signed decimal integer
e	Display argument using scientific notation
f	Display argument as floating point
o	Display argument as an octal integer
s	Display argument as a string
u	Display argument in unsigned decimal
x	Display argument in lowercase hexadecimal
X	Display argument in uppercase hexadecimal

Example:

- Suppose we want a color that has a triplet value of 65 red,127 green and 245 blue the hexadecimal value for these are
- Printf(“<font color=“#%X%X%X “Hello </font>”,65,127,245);

Precision setting

- In php we can also set the precision of the displayed result. For example: amounts of currency are usually displayed with only two digits of precision. However after calculation a value may have a greater precision than this e.g. $123.42/12, which result in $10.285).
- If we want to displayed only two digit of precision we can insert the string “.2” between the % symbol and the conversion specifier:

```
Printf(“the result is :$%.2f”,123.42/12);
Output is $10.29
```

String Padding

- We can also pad string to required lengths select different padding characters and even choose between left and right justification.

```
<php
echo “<pre>”;
$h=“house”;
Printf(“[%s]\n”,                    $h);
Printf(“[%10s]\n”,                  $h);
Printf(“[%-10s]\n”,                 $h);
Printf(“[%010s]\n”,                 $h);
?>
```

Sprintf function

- Using sprintf function we can send the output to another variable rather than to the browser.

  $hex string = sprintf("%X%X%X" ,65,127,245);

- Or we may wish to store the output in a variable

  $out= sprintf("the result is :$%.2f",123.42/12);

  echo $out;

Lecture-26
Forms in PHP(get &post Methods in form, get data form text boxes, password boxes, and hidden fields.

From: Ms. Suman Singh
Asst Professor(IT)

Introduction

- One of the most powerful features of PHP is the way it handles HTML forms.
- The basic concept that is important to understand is that any form element will automatically be available to your PHP scripts.
- Websites users interact with PHP and MYSQL is through the use of HTML forms.

To Build a form, We must have at least the following elements

1. An opening <form> and closing <form>
2. Submission type specifying either a GET or POST method.
3. One or more input fields.
4. Destination URL to which the form data is to be submitted.

Example : a simple HTML form with two input fields and a submit button:

```
<html>
   <body>

   <form action="welcome.php" method="post">
   Name: <input type="text" name="name"><br>
   E-mail: <input type="text" name="email"><br>
   <input type="submit">
   </form>

   </body>
   </html>
```

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
  <body>

  Welcome <?php echo $_POST["name"];
  ?><br>
  Your email address is: <?php echo
  $_POST["email"]; ?>

  </body>
  </html>
```

Example: A Simple HTML Form

```
<?php
echo <<<_end
<html>
<head>
<title> first form in PHP</title>
</head>
<form method="post" action="form.php">
Name: <input type ="text" name="name"/>
<input type="submit"/>
</form>
</body>
</html>
_end;
?.
```

Note:

- In above example if we enter a name and click on submit Query button nothing will happen other than the form being redisplayed.

- So for Retrieving data add some PHP code to process the data submitted by the form.

Example: simple form with PHP code for retrieving data

```
<?php
If(isset($_POST[‘name’]))
$name=$_POST[‘name’];
Else $name=“Not entered”;
echo <<<_end
    <html>
       <head>
            <title> first form in PHP</title>
    </head>
    <body>
    Your name is   :$name<br>
    <form method=“post” action=“form1.php”>
    Name: <input type =“text” name=“name”/>
    <input type=“submit”/>
    </form>
    </body>
    </html>
    _end;
    ?>
```

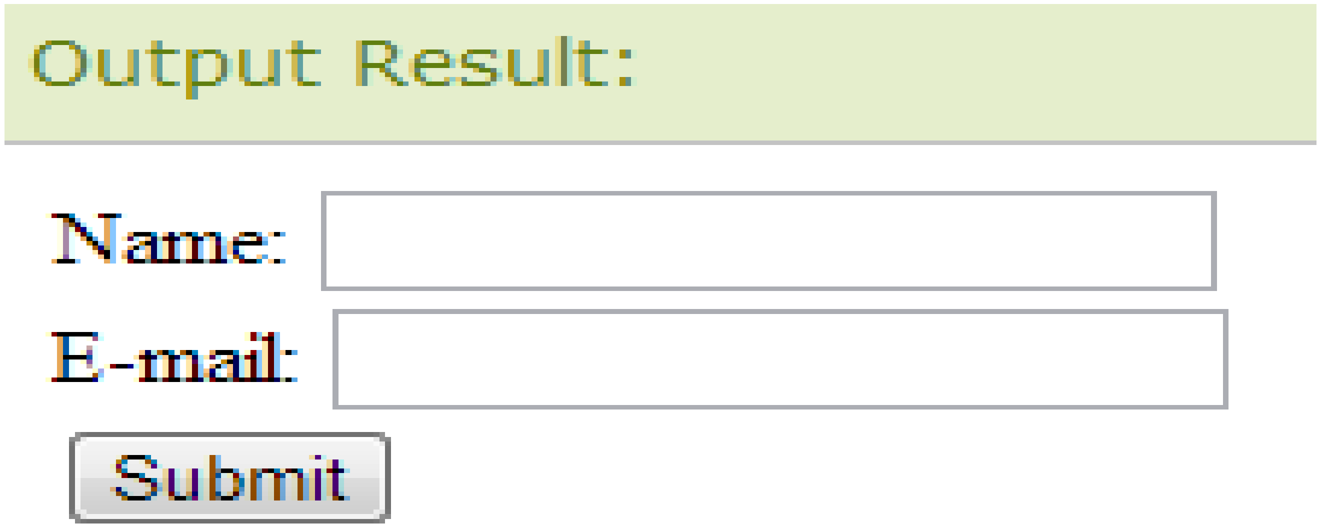
Note:

- $_POST associative array for field name submitted.
- $_POST associative array , which contains an element for each field in an HTML form.
- The input name used was name and the form method was POST , so element name of the $_POST array contains the value in $_POST['name'].
- PHP isset function is used to test whether $_POST['name] has been assign a value.
- If nothing was posted the program assigns the value " not entered"

The same result could also be achieved using the HTTP GET method:

```
<html>
   <body>
   <form action="welcome_get.php" method="get">
   Name: <input type="text" name="name"><br>
   E-mail: <input type="text" name="email"><br>
   <input type="submit">
   </form>
   </body>
   </html>
```

output


Output Result:

Name:

E-mail:

and "welcome_get.php" looks like this:

```
<html>
  <body>

  Welcome <?php echo $_GET["name"]; ?><br>
  Your email address is: <?php echo $_GET["email"];
  ?>

  </body>
  </html>
```

GET vs. POST

- Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
- Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and we can access them from any function, class or file without having to do anything special.
- $_GET is an array of variables passed to the current script via the URL parameters.
- $_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

- Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
- GET may be used for sending non-sensitive data.
- **Note:** GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

- Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.
- Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Default Values

- Sometime it convenient to offer our site visitors a default value in a web form .
- Example:

No. of year <input type =“text” name=“year” value”2”/>

VARIOUS FORM CONTROLS TAGS AVAILABLE IN HTML ARE:-

(1) Input types

- Html forms are very versatile and allow us to submit a wide range of different types of inputs ranging from text boxes and text areas to checkboxes, radio buttons and more.
- 1. Text boxes

  It accepts a wide range of alphanumeric text and other characters in a single-line box. General format of text box input is:

```
<input type="text" name="name" size="size"
   maxlength="length" value="value"/>
```

- Size parameter: Size parameter specifies the width of the box in characters of the current font
- Maxlength specifies the maximum number of characters that a user allowed to enter into the field.

Text area

- When we need to accept input of more than a single line of text , use a text area.

<textarea name="name" cols="width" rows="height" wrap="type">

</textarea>

- The <textarea> has its own tag and is not a subtype of the <input> tag . So there is a requirement to close </textarea> to end input
- Instead of default parameter we can have default text to display we must put the text before the closing textarea.

We can control how the text entered into the box will wrap (and how any such wrapping will be sent to the server) using the wrap parameter.

Type	Action
off	Text does not wrap and lines appear exactly as the user types them.
soft	Text wraps but is sent to the server as one long string without carriage returns and line feeds.
hard	Text wraps and is sent to the server in wrapped format with soft returns and line feeds.

Checkboxes

- When we want to offer a number of different options to a user from which he/she can select one or more items, checkboxes are used . The format is:

<input type=“checkbox” name=“name” value=“value” checked=“checked”/>

- ❑ If we use the check parameter the checkbox is already checked. When the browser is displayed.
- ❑ If we don’t include the parameter the box is shown unchecked.

Radio buttons

- <input type=“radio” name=“c” value=“1”>

- <select> element is used to create a drop-down list.

- <option> tags inside <select> element define the available options in the list.

  SYNTAX:

```
        <select>
                <option value="z">ABC</option>
                <option value="z">ABC</option>
        </select>
```

<optgroup> is used to group related options in a drop-down list.

SYNTAX:

```
<select>
<optgroup label=“Fruits”>
<option value=“a”>APPLE</option>
<option value=“b”>GUAVA</option>
</optgroup>
</optgroup>
<optgroup label=“Vegetables”>
<option value=“c”>POTATO</option>
<option value=“d”>TOMATO</option>
</optgroup>
</select>
```

Hidden fields

- Sometimes it is convenient to have hidden form fields so that we can keep track of the state of form entry.
- Example: we might wish to know whether a form has already been submitted. We can achieve this by adding some HTML such as

```
Echo '<input type="hidden" name="submitted" value="yes"/>'
```

- ❑ Php program receives the input the above line of code has not run , so there will be no field named submitted.
- ❑ The PHP program recreates the form , adding the input field then when the visitor resubmits the form the PHP program receives it with the submitted field set to "yes".
- ❑ The code simply check whether the field is present :

```
        If(isset($_POST['submitted']))
                {….
```

Submit Button

- <input type=“submit” value=“search”/>
- <input type=“image” name=“submit” src=“image.gif”/>

Lecture-27-28
PHP Form Validation

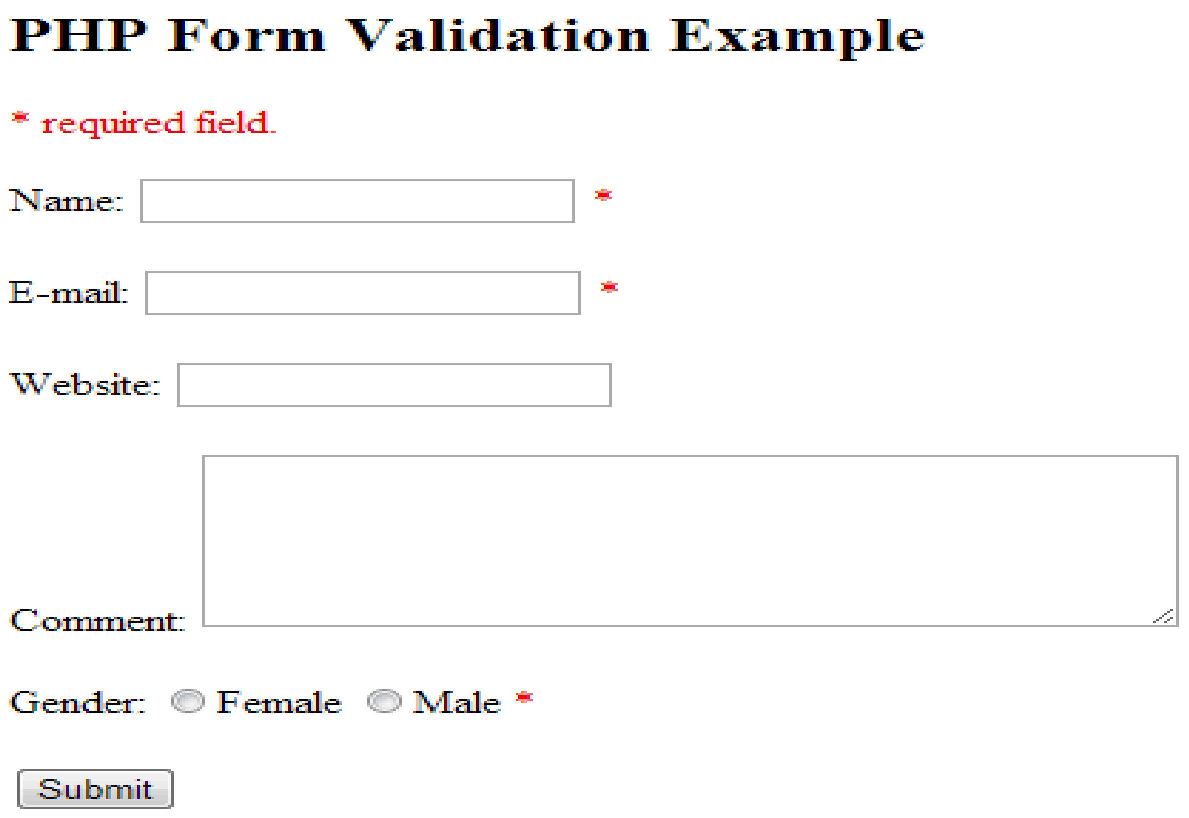
From: Ms. Suman Singh

Asst. Professor(IT)

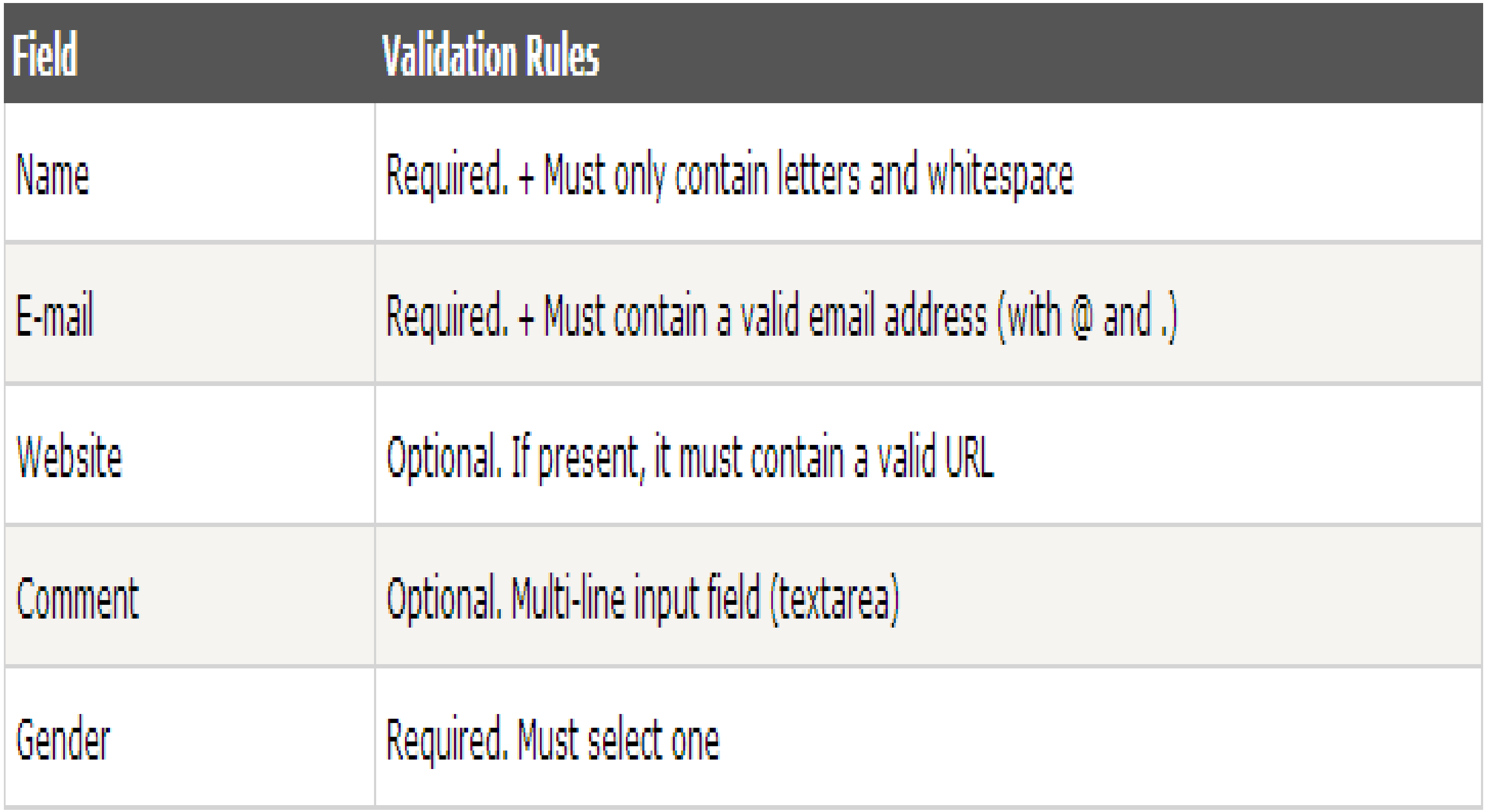
SECURITY when processing PHP forms

- Proper validation of form data is important to protect our form from hackers and spammers!

The HTML form contains various input fields: required and optional text fields, radio buttons, and a submit button



The validation rules for the form above are as follows:



Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

The Form Element

- <form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
- When the form is submitted, the form data is sent with method="post".
- The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
- So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

PHP Form Security

- The $_SERVER["PHP_SELF"] variable can be used by hackers.
- If PHP_SELF is used in our page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.
- **(Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.)**

- Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

  <form method="post" action="test_form.php">

- However, consider that a user enters the following URL in the address bar:

  http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

- In this case, the above code will be translated to:

<form method="post" action="test_form.php"/><script>alert('hacked')</script>

Note:

- This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.
- Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

- $_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.
- The form code should look like this:

  <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">

- **htmlspecialchars() function**

  -The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Note

- The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post"
  action="test_form.php/&quot;&gt;&lt;script&gt;alert('
  hacked')&lt;/script&gt;">
```

Validate Form Data With PHP

- The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.
- When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

  <script>location.href('http://www.hacked.com')</script>

  - this would not be executed, because it would be saved as HTML escaped code, like this: &lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;

  The code is now safe to be displayed on a page or inside an e-mail.

- We will also do two more things when the user submits the form:

- ❑ Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
- ❑ Remove backslashes (\) from the user input data (with the PHP stripslashes() function)
- ❑ The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

we can check each $_POST variable with the test_input() function, and the script look like this:

```
<?php
   // define variables and set to empty values
   $name = $email = $gender = $comment = $website = "";

   if ($_SERVER["REQUEST_METHOD"] == "POST")
   {
     $name = test_input($_POST["name"]);
     $email = test_input($_POST["email"]);
     $website = test_input($_POST["website"]);
     $comment = test_input($_POST["comment"]);
     $gender = test_input($_POST["gender"]);
   }

   function test_input($data)
   {
     $data = trim($data);
     $data = stripslashes($data);
     $data = htmlspecialchars($data);
     return $data;
   }
   ?>
```

Note:

- The start of the script, we check whether the form has been submitted using
- $_SERVER["REQUEST_METHOD"].
- If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated.
- If it has not been submitted, skip the validation and display a blank form.

How to create error messages if needed in form Validation

```
<?php
   // define variables and set to empty values
   $nameErr = $emailErr = $genderErr = $websiteErr = "";
   $name = $email = $gender = $comment = $website = "";

   if ($_SERVER["REQUEST_METHOD"] == "POST")
   {

    if (empty($_POST["name"]))
      {$nameErr = "Name is required";}
    else
      {$name = test_input($_POST["name"]);}

    if (empty($_POST["email"]))
      {$emailErr = "Email is required";}
    else
      {$email = test_input($_POST["email"]);}
```

```
 if (empty($_POST["website"]))
   {$website = "";}
 else
   {$website = test_input($_POST["website"]);}

 if (empty($_POST["comment"]))
   {$comment = "";}
 else
   {$comment = test_input($_POST["comment"]);}

 if (empty($_POST["gender"]))
   {$genderErr = "Gender is required";}
 else
   {$gender = test_input($_POST["gender"]);}
}
?>
```

```
form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">

Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
<label>Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">

</form>
```

PHP -Regular Expressions

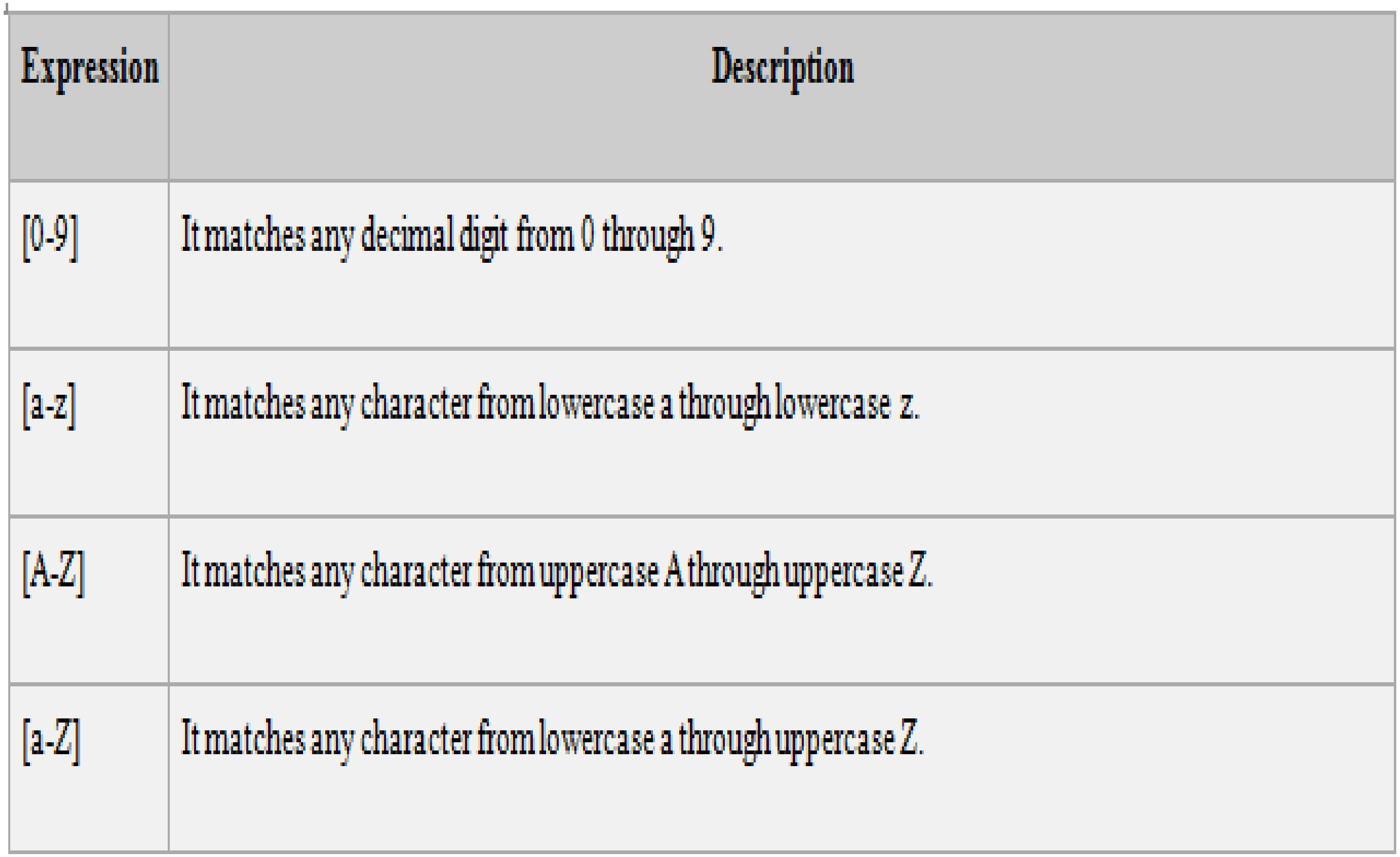
- Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.
- Using regular expression we can search a particular string inside a another string, we can replace one string by another string and we can split a string into many chunks.
- PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. We can use any of them based on our comfort.
  - POSIX Regular Expressions
  - PERL Style Regular Expressions

POSIX Regular Expressions:

- The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.

Brackets

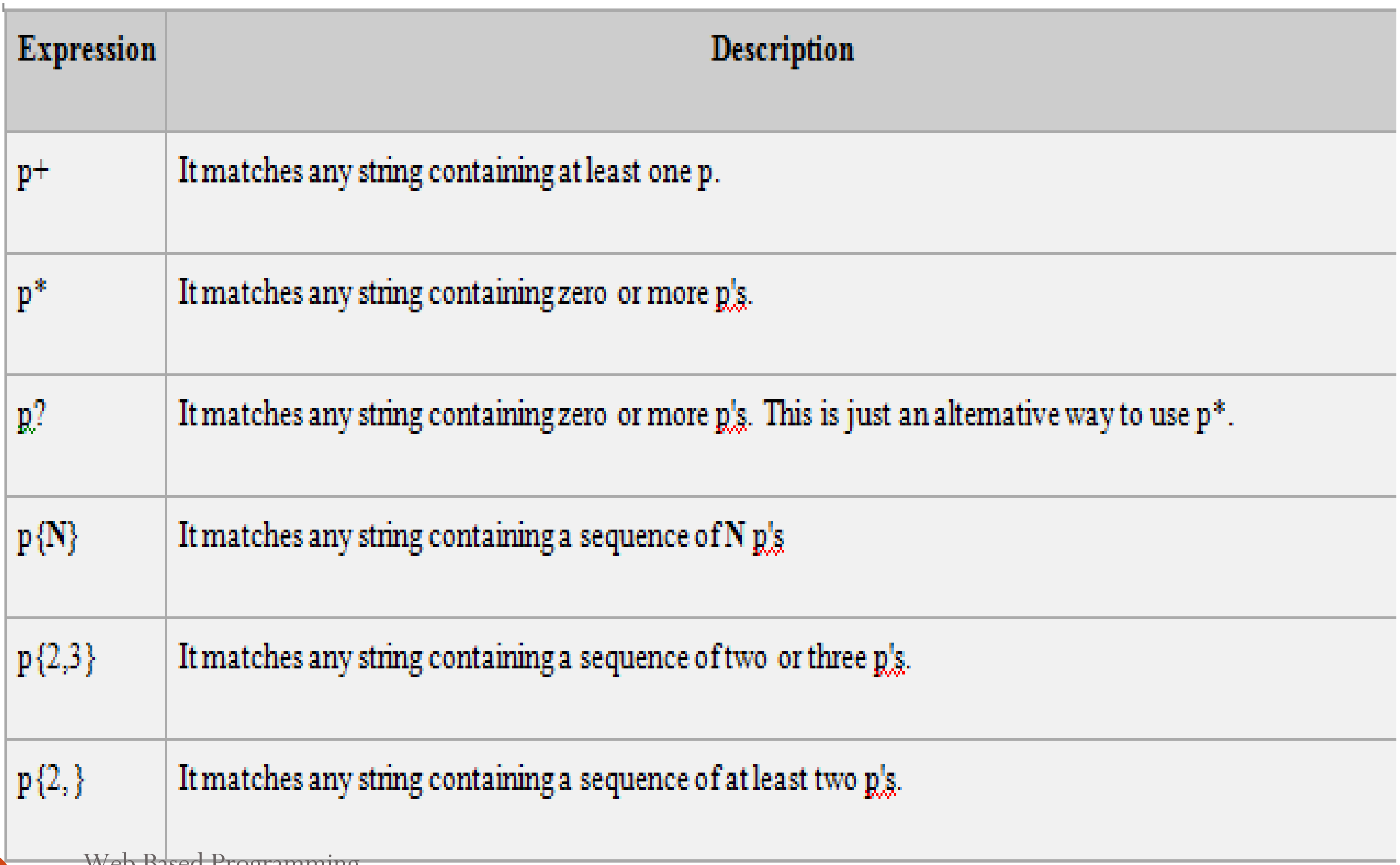
Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.



Expression	Description
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

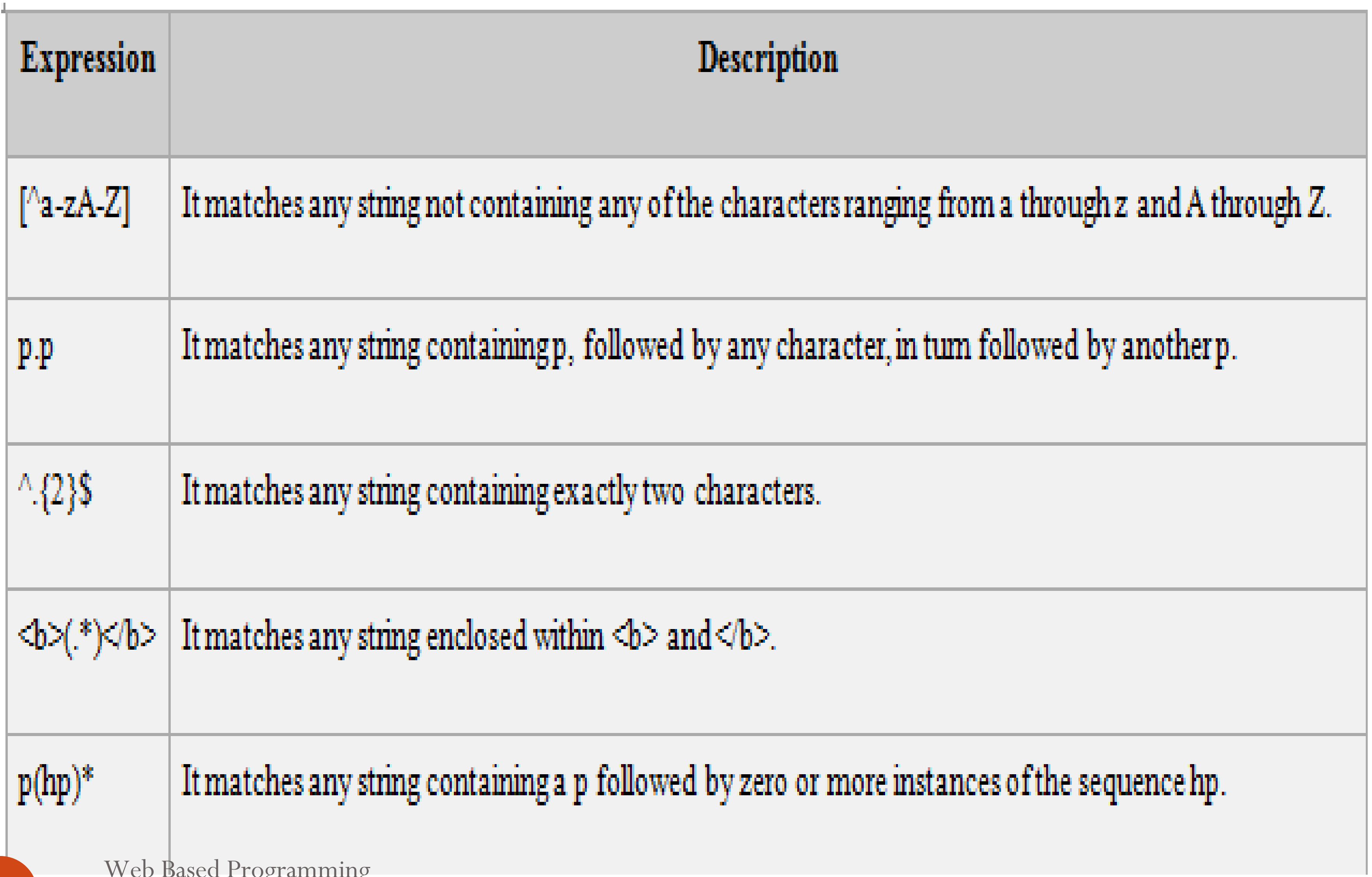
Quantifiers:

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific notation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.



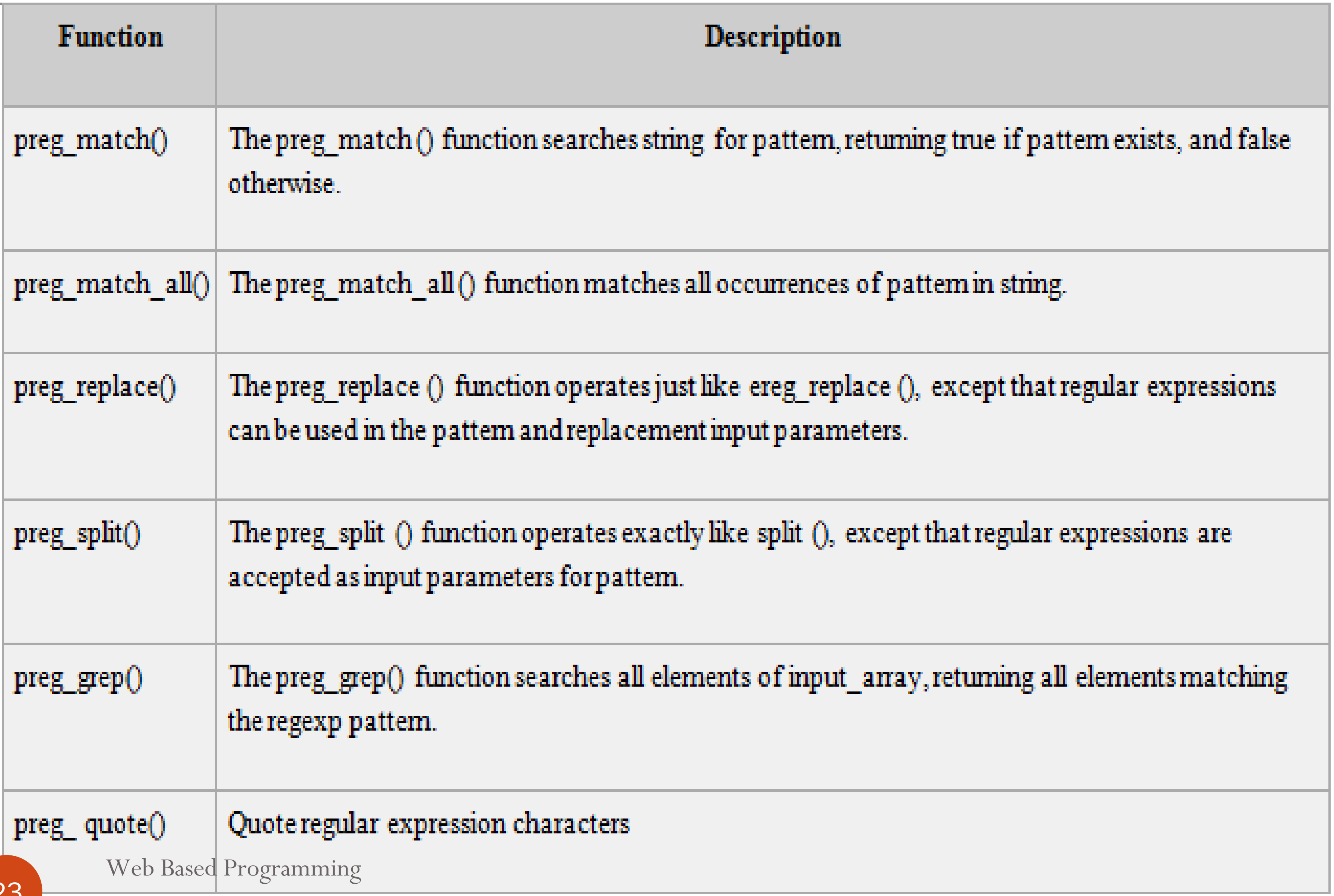
Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing zero or more p's. This is just an alternative way to use p*.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.

Examples:



Expression	Description
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^.{2}$	It matches any string containing exactly two characters.
<b>(.*)</b>	It matches any string enclosed within <b> and </b>.
p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence hp.

PHP's Regular expression Compatible Functions



Function	Description
preg_match()	The preg_match () function searches string for pattern, returning true if pattern exists, and false otherwise.
preg_match_all()	The preg_match_all () function matches all occurrences of pattern in string.
preg_replace()	The preg_replace () function operates just like ereg_replace (), except that regular expressions can be used in the pattern and replacement input parameters.
preg_split()	The preg_split () function operates exactly like split (), except that regular expressions are accepted as input parameters for pattern.
preg_grep()	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
preg_ quote()	Quote regular expression characters

PHP - Validate Name

- if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name))
  {
  $nameErr = "Only letters and white space
allowed";
  }
```

(The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.)

PHP - Validate E-mail

- To check if an e-mail address syntax is valid. If the e-mail address syntax is not valid, then store an error message:

```
$email = test_input($_POST["email"]);
if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email))
  {
  $emailErr = "Invalid email format";
  }
```

PHP - Validate URL

- To check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
  if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/ i“ , $website))
    {
    $websiteErr = "Invalid URL";
    }
```

Example:

```
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST")
{
   if (empty($_POST["name"]))
     {$nameErr = "Name is required";}
   else
     {
     $name = test_input($_POST["name"]);
     // check if name only contains letters and whitespace
     if (!preg_match("/^[a-zA-Z ]*$/",$name))
       {
       $nameErr = "Only letters and white space allowed";
       }
     }
```

```
if (empty($_POST["email"]))
     {$emailErr = "Email is required";}
   else
     {
     $email = test_input($_POST["email"]);
     // check if e-mail address syntax is valid
     if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email))
       {
       $emailErr = "Invalid email format";
       }
     }

   if (empty($_POST["website"]))
     {$website = "";}
   else
     {
     $website = test_input($_POST["website"]);
     // check if URL address syntax is valid (this regular expression also allows
dashes in the URL)
     if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i", $website))
       {
       $websiteErr = "Invalid URL";
       }
     }
```

```
if (empty($_POST["comment"]))
      {$comment = "";}
    else
      {$comment = test_input($_POST["comment"]);}

    if (empty($_POST["gender"]))
      {$genderErr = "Gender is required";}
    else
      {$gender = test_input($_POST["gender"]);}
  }

  function test_input($data)
  {
     $data = trim($data);
     $data = stripslashes($data);
     $data = htmlspecialchars($data);
     return $data;
  }
  ?>
```

```
<h2>PHP Form Validation Example</h2>
  <p><span class="error">* required field.</span></p>
  <form method="post" action="<?php echo
  htmlspecialchars($_SERVER["PHP_SELF"]);?>">
    Name: <input type="text" name="name" value="<?php echo $name;?>">
    <span class="error">* <?php echo $nameErr;?></span>
    <br><br>
    E-mail: <input type="text" name="email" value="<?php echo $email;?>">
    <span class="error">* <?php echo $emailErr;?></span>
    <br><br>
    Website: <input type="text" name="website" value="<?php echo $website;?>">
    <span class="error"><?php echo $websiteErr;?></span>
    <br><br>
    Comment: <textarea name="comment" rows="5" cols="40"><?php echo
  $comment;?></textarea>
    <br><br>
    Gender:
    <input type="radio" name="gender" <?php if (isset($gender) && $gender=="female") echo
  "checked";?>  value="female">Female
    <input type="radio" name="gender" <?php if (isset($gender) && $gender=="male") echo
  "checked";?>  value="male">Male
    <span class="error">* <?php echo $genderErr;?></span>
    <br><br>
    <input type="submit" name="submit" value="Submit">
  form>
```

```
<?php
   echo "<h2>Your Input:</h2>";
   echo $name;
   echo "<br>";
   echo $email;
   echo "<br>";
   echo $website;
   echo "<br>";
   echo $comment;
   echo "<br>";
   echo $gender;
   ?>

   </body>
   </html>
```

Identifying Browser & Platform

- PHP creates some useful **environment variables** that can be seen in the phpinfo.php page that was used to setup the PHP environment.
- One of the environment variables set by PHP is **HTTP_USER_AGENT** which identifies the user's browser and operating system.
- PHP provides a function getenv() to access the value of all the environment variables. The information contained in the HTTP_USER_AGENT environment variable can be used to create dynamic content appropriate to the browser.

Following example demonstrates how you can identify a client browser and operating system.

```
<html>
<body>
<?php
   $viewer = getenv( "HTTP_USER_AGENT" );
   $browser = "An unidentified browser";
   if( preg_match( "/MSIE/i", "$viewer" ) )
   {
      $browser = "Internet Explorer";
   }
   else if(  preg_match( "/Netscape/i", "$viewer" ) )
   {
      $browser = "Netscape";
   }
```

```
else if(  preg_match( "/Mozilla/i", "$viewer" ) )
   {
      $browser = "Mozilla";
   }
   $platform = "An unidentified OS!";
   if( preg_match( "/Windows/i", "$viewer" ) )
   {
      $platform = "Windows!";
   }
   else if ( preg_match( "/Linux/i", "$viewer" ) )
   {
      $platform = "Linux!";
   }
   echo("You are using $browser on $platform");
?>
</body>
</html>
```

Using HTML Forms

- The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts.

```
<?php
 if( $_POST["name"] || $_POST["age"] )
 {
   echo "Welcome ". $_POST['name']. "<br />";
   echo "You are ". $_POST['age']. " years old.";
   exit();
 }
?>
<html>
<body>
 <form action="<?php $_PHP_SELF ?>" method="POST">
 Name: <input type="text" name="name" />
 Age: <input type="text" name="age" />
 <input type="submit" />
</form></body></html>
```

The PHP default variable **$_PHP_SELF** is used for the PHP script name and when we click "submit" button then same PHP script will be called and will produce following result:

Browser Redirection

- The PHP **header()** function supplies raw HTTP headers to the browser and can be used to redirect it to another location. The redirection script should be at the very top of the page to prevent any other part of the page from loading.
- The target is specified by the **Location:** header as the argument to the **header()** function. After calling this function the **exit()** function can be used to halt parsing of rest of the code.

Example:

```
<?php
 if( $_POST["location"] )
 {
   $location = $_POST["location"];
   header( "Location:$location" );
   exit();
 }
?>
```

```
<html>
<body>
  <p>Choose a site to visit :</p>
  <form action="<?php $_PHP_SELF ?>" method="POST">
  <select name="location">
    <option value="http://w3c.org">
        World Wise Web Consortium
    </option>
    <option value="http://www.google.com">
        Google Search Page
    </option>
  </select>
  <input type="submit" />
  </form>
</body>
</html>
```

Lecture-29
Query Strings: Quoting String Constants, Printing Strings, Cleaning Strings, Comparing String

Query string

- In the World Wide Web, a **query string** is the part of a uniform resource locator (URL) that contains data to be passed to web applications.
- When a web page is requested via the Hypertext Transfer Protocol, the server locates a file in its file system based on the requested URL.
- This file may be a regular file or a program. In the second case, the server may (depending on its configuration) run the program, sending its output as the requested page.
- The query string is a part of the URL which is passed to the program. Its use permits data to be passed from the HTTP client (often a web browser) to the program which generates the web page.

- A typical URL containing a query string is as follows:
- http://server/program/path/?query_string
- When a server receives a request for such a page, it may run a program, passing the query_string unchanged to the program. The question mark is used as a separator and is not part of the query string.
-

- field1=value1&field2=value2&field3=value3..
- The query string is composed of a series of field-value pairs.
- Within each pair, the field name and value are separated by an equals sign, '='.
- The series of pairs is separated by the ampersand, '&' (or semicolon, ';' for URLs embedded in HTML).

Example: query String

```
<?php
$a=10;
$b=20;
header("location: query.php?var=$a&var1=$b");
?>
```

Query.php

```
<?php
$x=$_REQUEST["var"];
$y=$_REQUEST["var1"];
echo $x;
echo $y;
?>
```

Lecture-30
PHP Cookies

Maintaining User State: using cookies in php(setting, accessing and destroying cookies)

Definition

- Cookies provide a way for a web application to store information in the user’s web browser and retrieve it every time the user request a page.
- Or
- Cookies are text files stored on the client computer and they are kept of use tracking purpose.
- Cookies also known as HTTP cookies or browser cookies .

Introduction

- A cookie is a name/value pair that is stored in a browser.
- On the server a web application creates a cookie and sends it to the browser. On the client the browser saves the cookie and sends it back to the server every time it accesses a page from that server.
- By default , cookies only last until the user close his/her web browser. However , cookies can be set to persist in the user’s browser for up to three years.

- Some users disables cookies in their browsers. As a result we can not always count on all user having their cookies enabled.
- Browsers generally accepts only 20 cookies from each site and 300 cookies total. In addition they can limit each cookie to 4 kilobytes.
- A cookie can be associated with one or more subdomain names.

Uses for cookies

- To allow users to skip login and registration forms that gather data like username, password, address or any data.
- To customize pages that display information like weather report ,sports scores and stock quotations.
- To focus advertising like banner ads that target the user’s interests.

Note:

- A common misconception is that cookies are harmful. Cookies consists only of plain text , they can not directly modify user's computer create pop-up ads, generate spam or steal files.

Examples of cookies

- PHPSESSID=D1F1523456666767777
- User_id=87
- Email=abc@gmail.com
- Username=abc
- Passwordcookie=xyz

- Cookies are usually set in a HTTP header(although java script can also set a cookie directly on a browser). A php script that sets a cookie might send headers that look like

HTTP/ 1.1 2000k

Date:

Server:

Set: cookie: name=“”expire=“”

How to 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);
```

Parameters of the setcookie function

- **Name -** This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value -**This sets the value of the named variable and is the content that you actually want to store.
- **Expiry -** This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path -**This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain -** This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security -** This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Note:

- If we set the $expire parameter to 0 the cookie only exists until the user closes the browser, this is called a pre-session cookie.
- If we set the $expire parameter to a date up to three years from the current date. In that case cookie stays in the browser until the expiration date this is called a persistent cookie.

Example 1

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

```
        <?php
        setcookie("user", "Alex ", 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: to set cookie

```
<?php
    setcookie("name", "John ", time()+3600, "/","", 0);
    setcookie("age", "36", time()+3600, "/", "",  0);
  ?>
  <html>
  <head>
  <title>Setting Cookies with PHP</title>
  </head>
  <body>
  <?php echo "Set Cookies"?>
  </body>
  </html>
```

Example 2

- We 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",
"Alex", $expire);
?>
<html>
 ....
```

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

How to Retrieve a Cookie Value?

- 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>
```

Example: **Accessing Cookies with PHP**

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
echo $_COOKIE["name"]. "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"]. "<br />";

echo $_COOKIE["age"] . "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"] . "<br />";
?>
</body>
</html>
```

Example: Accessing Cookie using isset function

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
  if( isset($_COOKIE["name"]))
    echo "Welcome " . $_COOKIE["name"] . "<br />";
  else
    echo "Sorry... Not recognized" . "<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);
?>
```

Example: Delete a cookie

```
<?php
  setcookie( "name", "", time()- 60, "/","", 0);
  setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
<head>
<title>Deleting Cookies with PHP</title>
</head>
<body>
<?php echo "Deleted Cookies" ?>
</body>
</html>
```

What if a Browser Does NOT Support Cookies?

- If our application deals with browsers that do not support cookies, we will have to use other methods to pass information from one page to another in our application.

"welcome.php" when the user clicks on the "Submit" button:

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name">
Age: <input type="text" name="age">
<input type="submit">
</form>

</body>
</html>
```

Retrieve the values in the "welcome.php" file like this:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?>.<br>
You are <?php echo $_POST["age"]; ?> years old.

</body>
</html>
```

Enable or disable cookies

- To test how our application behaves if a user has cookies disabled we can disable cookies in our browser.
- To test how our application behaves under normal circumstances , we can enable cookies in our browser.

To enable or disable cookies in Firefox 3.6

1. Open the tool menu and select the option command
2. Click on the privacy tab.
3. Use the accepts cookies from sites" check box to enable or disable cookies.

To enable or disable cookies in Internet Explorer 8

1. open the tools menu and select the internet options command.
2. Click the privacy tab.
3. Use the slider control to enable or disable cookies. To disable cookies , set the security level to “Block all cookies”. To enable cookies , click the default button to return to default privacy settings.

To reset default security settings in in Internet Explorer 8

1. Open the Tools menu and select the Internet Options command.
2. Click the security tab.
3. If not disabled, click the “Reset all zones to default level” button.

Lecture-31
Maintaining User State: Sessions (starting, ending and session security), application

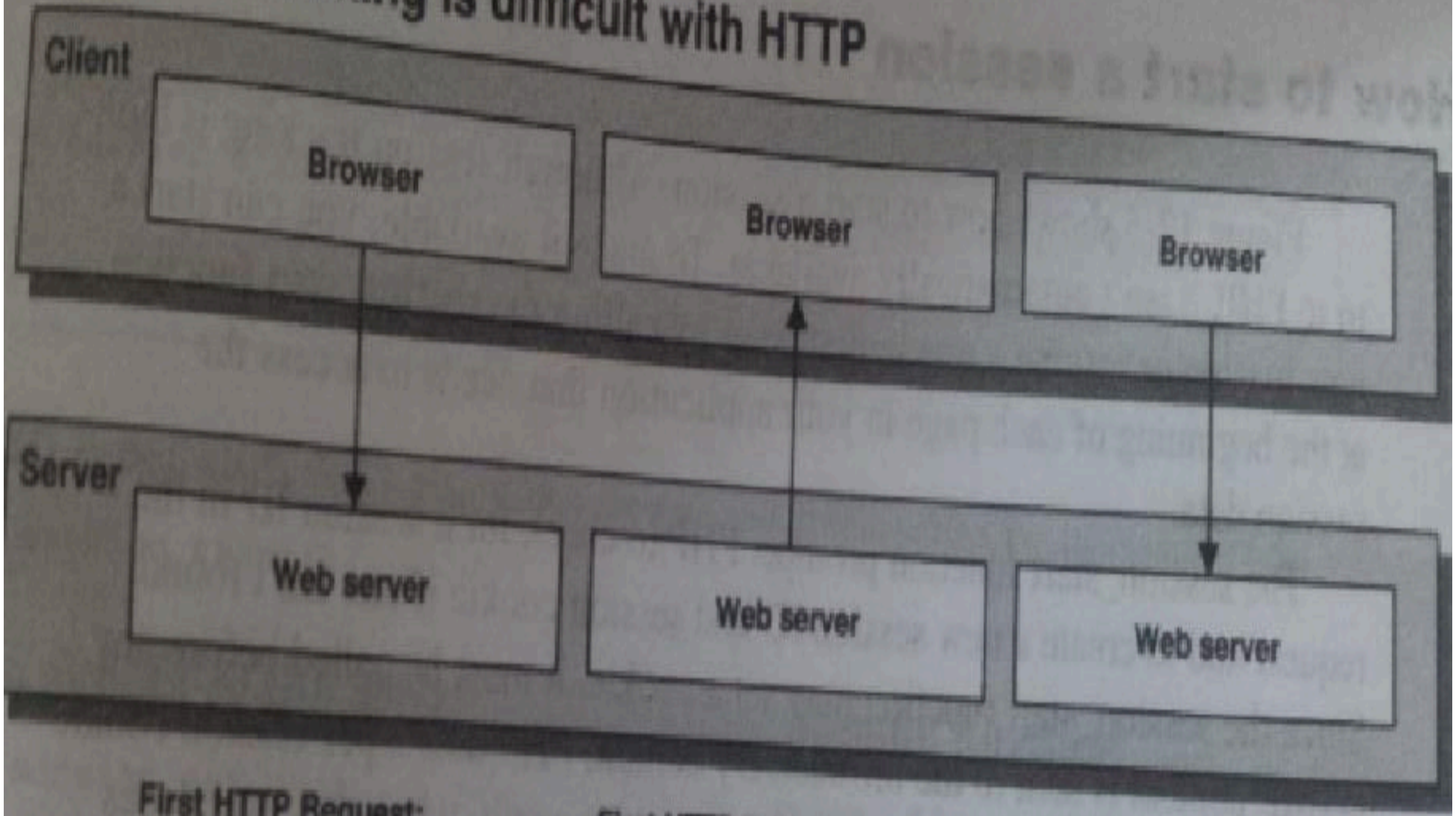
Introduction

- Keeping track of users as they move around a web site is known as session tracking.
- A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

Session Tracking Difficulty with HTTP

- HTTP is a stateless protocol. Once a browser makes a request, it drops the connection to the server. To maintain state a web application uses session tracking.

acking is difficult with HTTP



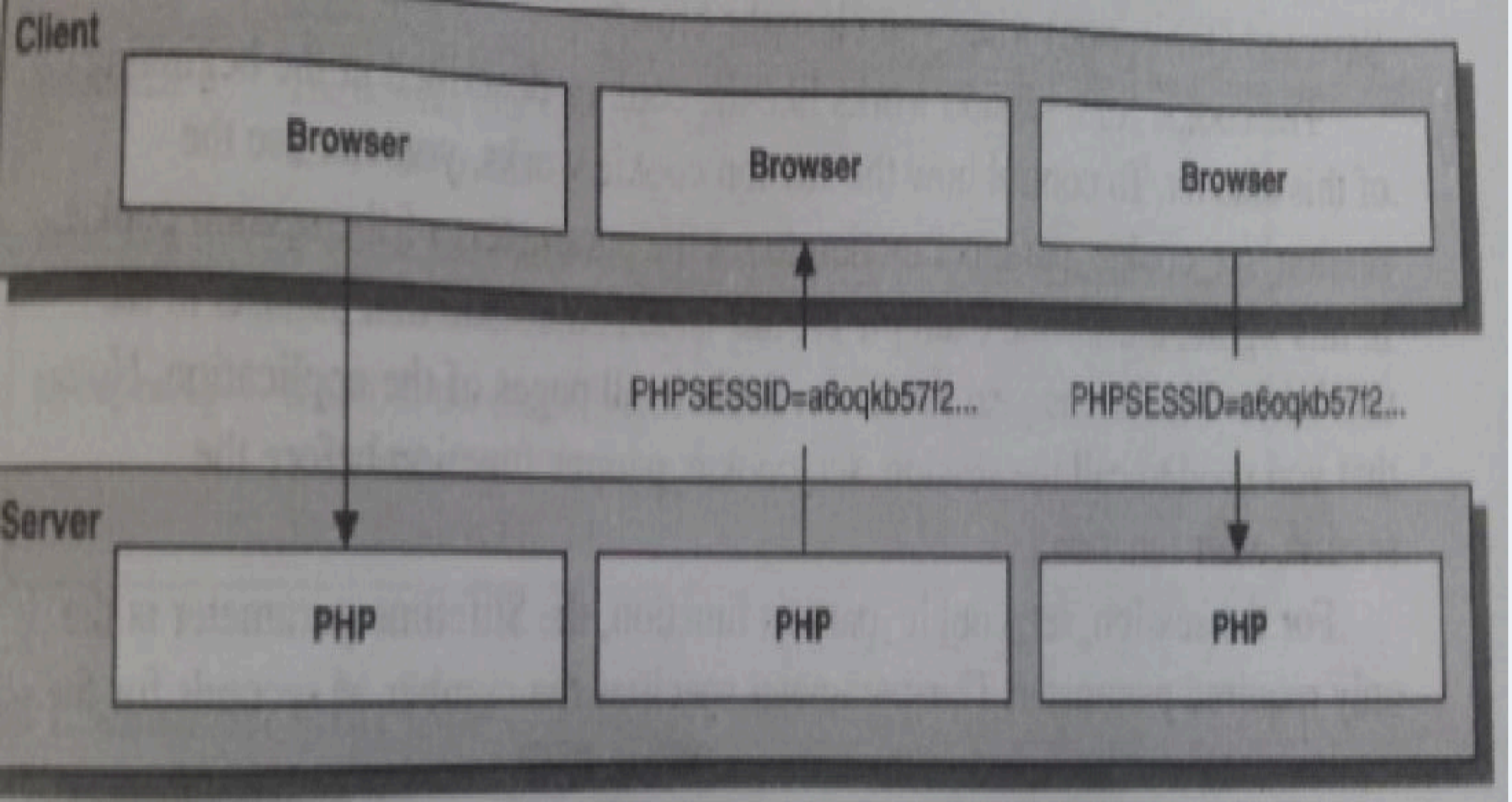
First HTTP Request:
The browser requests a page.

First HTTP Response:
The server returns the requested page and drops the connection.

Following HTTP Requests:
The browser requests a page. The web server has no way to associate the browser with its previous request.

Session tracking works in php

- By default , PHP uses a cookie to store a session ID in each browser. Then the browser passes the cookie to server with each request.
- To provides session tracking when cookie when cookies are disabled in the browser , we can use URL encoding to store the session ID in the URL for each pages of an application.



First HTTP Request:
The browser requests a PHP page. PHP creates a session and assigns it an ID.

First HTTP Response:
The server returns the requested page and the ID for the session as a cookie.

Following HTTP Requests:
The browser requests a PHP page and sends the session ID cookie. PHP uses the session ID to associate the browser with its session.

PHP Session Variables

- When you are working with an application, we open it, do some changes and then we 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 a PHP 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

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

-
  ```
  <?php
  session_start();
  // store session data
  $_SESSION['views']=1;
  ?>

  <html>
  <body>

  <?php
  //retrieve session data
  echo "Pageviews=". $_SESSION['views'];
  ?>

  </body>
  </html>
  ```

- In the example below, we create a simple page-views counter. 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'];
?>
```

Example: start a session

```
<?php
  session_start();
  if( isset( $_SESSION['counter'] ) )
  {
    $_SESSION['counter'] += 1;
  }
  else
  {
    $_SESSION['counter'] = 1;
  }
  $msg = "You have visited this page ".  $_SESSION['counter'];
  $msg .= "in this session.";
?>
<html>
<head>
<title>Setting up a PHP session</title>
</head>
<body>
<?php echo ( $msg ); ?>
</body>
</html>
```

Destroying a Session

- If we wish to delete some session data, you can use the unset() or the session_destroy() function.
- The unset() function is used to free the specified session variable:

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

- We 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.

Sessions without cookies:

- There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.
- Alternatively, we can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string. Thus, we can embed it unconditionally into URLs.
-

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
<?php
   session_start();

   if (isset($_SESSION['counter'])) {
      $_SESSION['counter'] = 1;
   } else {
      $_SESSION['counter']++;
   }
?>
   $msg = "You have visited this page ".  $_SESSION['counter'];
   $msg .= "in this session.";
   echo ( $msg );
<p>
To continue  click following link <br />
<a  href="nextpage.php?<?php echo htmlspecialchars(SID); >">
</p>
```

Manage a session

- We can use several function to manage a session.

Function	Description
Session_name()	Get the name of the session cookie. The default is PHPSESSID
Session_id()	If the parameter is not specified this function gets the current session id. If no session exists this function gets the empty string.
Session_write_close()	End the current session and saves session data . This function is only needed in special case like redirects.
Session_regenerate_id()	Create a new session ID for the current session .Returns True if successful and False otherwise .This function can be used to help prevent session hijacking.

HTTP Authentication

- Http authentication uses the web server to manage user and passwords for the application.
- To use HTTP authentication, PHP sends a header request asking to start an authentication dialog with the browser.
- When user enter URL into browser or visit via a link an "Authentication Required" prompt pops up require sting two fields: username and password

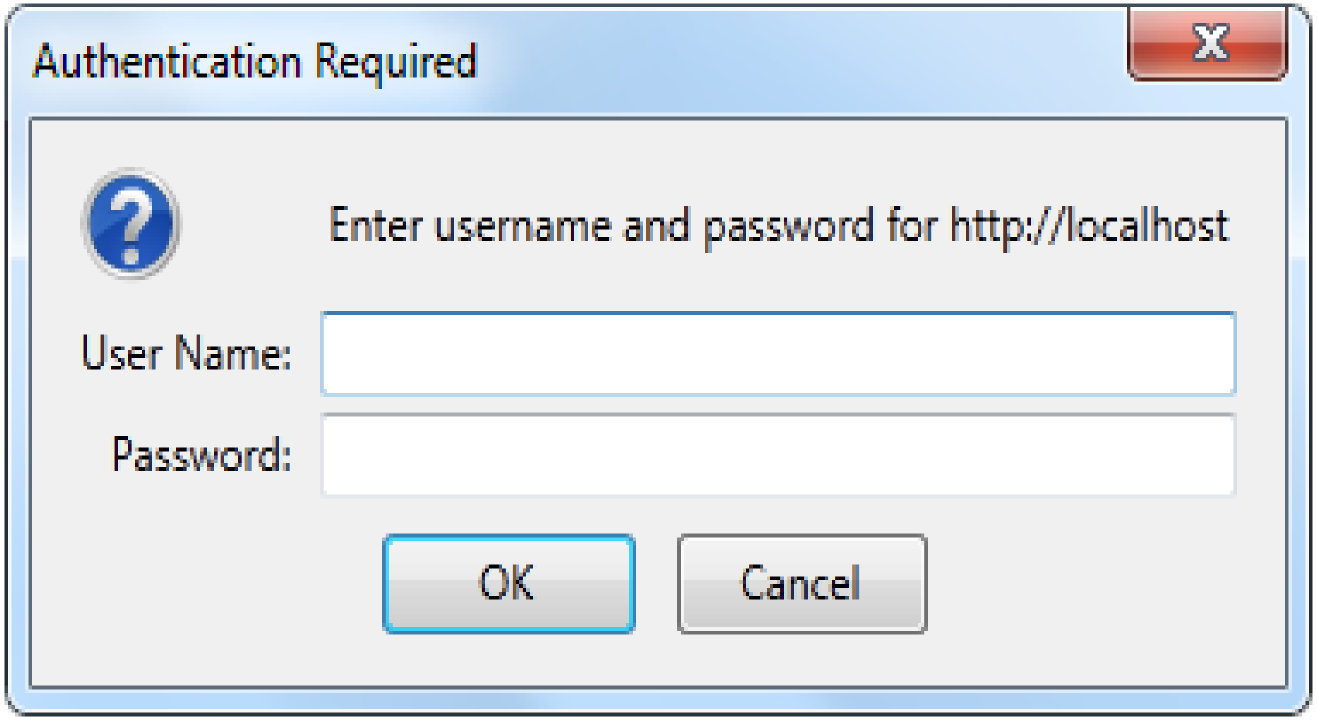
Example: PHP authentication with input checking

```
<?php
$username="admin";
$password="letmein";
if(isset ($_SERVER["PHP_AUTH_USER"])&& isset ($_SERVER["PHP_AUTH_PW"]))
{


if($_SERVER["PHP_AUTH_USER"]==$username  &&
    $_SERVER["PHP_AUTH_PW"]==$password)


echo "YOU ARE  NOW LOGGED IN";
else
die("invalid username/ password combination");
}
else
{
header('WWW-authenticate: basic realm="restricted section"');
header("HTTP/1.0 401 Unauthorized");
die("please  enter your username and password");
}
```

Output


Authentication Required

Enter username and password for http://localhost

User Name:

Password:

Storing usernames and passwords

- MySQL is the natural way to store username and passwords.
- Username and password not stored as clear text.
- To store password a md5 function is used.
- In md5 we pass a string to hash and return a 32 character hexadecimal number.
- Example: $token=md5(“suman”);

Lecture -32-33
PHP File Handling

Working with Files: File Handling(file exist, creating a file ,reading a file, copying ,writing deleting a file ,
Getting information on Files updating files, locking files ,reading an entire file

PHP Files

- Data that used by website is stored in a database.
- Some of the data might be stored in files, then we can use PHP to read data from the files and write data to the files.

Functions to determine if a file or directory exists.

1. is_file($path) : Return true if $path exists and is a file.
2. is_dir($path) : Return True if $path exists and is a directory.
3. file_exists($path) : Return True if $path exists and is either a file or a directory.

Function to get the current directory

getcwd()   : returns a string that specifies the current working directory.

Function to get a directory listing

scandir($path) : returns an array containing a list of the files and directory in $path if $path is valid directory name . Otherwise, it return False.

Constant that contains the correct path separator

DIRECTORY_SEPARATOR	: A backslash on windows servers or a forward slash on Mac OS and Linux servers.

Example: Display a directory listing

```
<html>
<body>
<?php
$path=getcwd();
$items=scandir($path);
echo"<p>contents of $path</p>";
echo"<ul>";
foreach($items as $item)
{
echo "<li>".$item."</li>";
}
echo "</ul>";
?>
</body>
</html>
```

Example: Display the files from a directory listing

```
<html>
<body>
<?php
$path=getcwd();
$items=scandir($path);
$files=array();
foreach($items as $item)
{
$item_path = $path. DIRECTORY_SEPARATOR .$item;
if(is_file($item_path))
{
$files[] = $item;
}
}
echo"<p>Files in $path</p>";
echo"<ul>";
foreach($files as $file)
{
echo "<li>".$file."</li>";
}
echo "</ul>";
?>
</body>
</html>
```

Opening and Closing Files

- The PHP **fopen()** function is used to open a file.
- It requires two arguments stating first the file name and then mode in which to operate.
- fopen(“filename” , “mode”)
- Example:

```
<html>
<body>
<?php
        $file=fopen(“emp.txt","r")
        or
        exit("Unable to    open file!");
?>
</body>
</html>
```

Closing a File: The fclose() function is used to close an open file:

Example:

```
<?php
$file = fopen("emp.txt","r");
$text=fread($file,5);
fclose($file);
echo $text;
?>
```

Files modes can be specified as one of the six options in this table.

Mode	Purpose
r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
w+	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
a+	Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

Note:

- If an attempt to open a file fails then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.
- After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

- Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.
- The file's length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.
- The steps required to read a file with PHP.
  - Open a file using **fopen()** function.
  - Get the file's length using **filesize()** function.
  - Read the file's content using **fread()** function.
  - Close the file with **fclose()** function.

Example: The following example assigns the content of a text file to a variable then displays those contents on the web page.

```
<html>
<head>
<title>Reading a file using PHP</title>
</head>
<body>

<?php
$filename = "C:\wamp\www\suman\emp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
  echo ( "Error in opening file" );
  exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );

fclose( $file );

echo ( "File size : $filesize bytes" );
echo ( "<pre>$filetext</pre>" );
?>
</body>
</html>
```

PHP provides functions for read an entire file into a string variable and write a string variable to a file. Three functions to read an entire file

1. file($name) : return an array with each element containing once line from the file.
2. file_get_contents($name) :returns the contents of the file as a string.
3. readfile($name) : reads a file and echoes is to the web page.

Example: read text from a file

```
<?php
$text=file_get_contents("emp.txt");
$text=htmlspecialchars($text);
echo"<div>".$text."</div>";
?>
```

Writing a file

- A new file can be written or text can be appended to an existing file using the PHP **fwrite()**function.
- fwrite() function requires two arguments specifying a **file pointer** and the string of data that is to be written.
- Optionally a third integer argument can be included to specify the length of the data to write.
- If the third argument is included, writing would will stop after the specified length has been reached.

Example:

```
<?php
$filename = "C:\wamp\www\suman\emp.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
   echo ( "Error in opening new file" );
   exit();
}
fwrite( $file, "This is  a simple test\n" );
fclose( $file );
?>
<html>
<head>
<title>Writing a file using PHP</title>
</head>
```

```
<body>
 <?php
 if( file_exist( $filename ) )
{
 $filesize = filesize( $filename );
  $msg = "File  created with name $filename ";   $msg .=
    "containing $filesize bytes";
  echo ($msg );
}
else
{
 echo ("File $filename does not exit" );
}
?>
</body>
</html>
```

Function to write an entire file

- file_put_contents($name, $data)   : Writes the specified data string to the specified filename.
- Example:

```
<?php
$text="this is line 1.\nThis is line 2.\n";
file_put_contents("emp.txt",$text);
echo "text has been written";
?>
```

Read and write part of a file

- For large files we may need to read and write portions of a file at a time since this can use large amounts of internal memory.
- So we can use the functions to read and write portions of a file within a loop until the entire file is processed.

<u>Function that read from and write to a file</u>

fread($filename, $length)	: Read up to the specified number of bytes from the specified file.
fgets($filename)	: Reads a line from the specified file.
fwrite($filename, $data)	:Write the specified string data to the specified file.

Modes used when opening a file with the fopen function

Mode	Description
“rb”	Open the file for reading. If the file does not exist, fopen returns FALSE.
“wb”	Opens the file for writing. If the file exists, the existing data is deleted. If the file does not exist ,it is created.
“ab”	Opens the file for writing, if the file exists , the new data is appended. If the file does not exist, it is created.
“xb”	Creates a new file for writing. If the file exist ,fopen returns FALSE.

Example: Read from a file

```
<?php
$file=fopen("emp.txt", "rb");
$name=" ";
while(!feof($file))
{
$name=fgets($file);
if($name=== false)
continue;
$name =trim($name);
if(strlen($name)==0 ||substr($name,0,1)=="#")
continue;
$names="";
$names.="<div>".$name."</div?\n";
}
fclose($file);
echo $names;
?>
```

Example: Write to a file

```
<?php
$path=getcwd();
$items= scandir($path);
$file=fopen("emp.txt", "wb");
foreach($items as $item)
{
$item_path =$path. DIRECTORY_SEPARATOR .$item;
if(is_dir($item_path))
{
fwrite($file, $item ."\n");
}
}
fclose($file);
?>
```

Read and write Comma –separated value

- When we work with text files ,we may use a type of text file called a comma-separated value(CSV) file.
- This type of file are exported from a spreadsheet or database table and it can be imported into a spreadsheet or database table.
- In a CSV file each line represents one record and each record contains value that are separated by commas.
- If a field within a record contains a comma , double quotes or line break the field must surrounded by double quotes.
- Any double quotes inside the field must be escaped by another set of double quotes.

Functions that write to a file

- fgetcsv($file) :Reads in a line of comma-separated values and returns them in any array.
- fputcsv($file , $array) :writes the specified array to the specified file as a line of comma- separated values.

Copy, rename and delete a file

- Functions to copy, rename and delete files

1. **Copy($oldname ,$newname)** :copies the old filename to the new file name .If successful return TRUE.
2. **Rename($oldname , $newname**) :rename the old filename to the new filename. If successful , returns TRUE.
3. **Unlink($name)** :Deletes the specified file .If successful , returns TRUE.

Example: Copy a file

```
<?php
$name1="emp.txt";
$name2="emp1.txt";
if(file_exists($name1))
{
$success=copy($name1,$name2);
if($success)
{
echo"<div> file was copied.</div>";
}
}
?>
```

Example: rename a file

```
<?php
$name1="emp1.txt";
$name2="emp2.txt";
if(file_exists($name1))
{
$success=rename($name1,$name2);
if($success)
{
echo"<div> file was renamed.</div>";
}
}
?>
```

Example: delete a file

```
<?php
$name="emp2.txt";
if(file_exists($name))
{
$success=unlink($name);
if($success)
{
echo"<div> file was deleted.</div>";
}
}
?>
```

Uploading files

- To upload a file an HTML for we can code

```
<form action="upload.php" method="post"
   enctype="multipart/form-date">
<input type ="file" name="file1"><br/>
<input type="submit" value="upload">
</form>
```

Elements of the nested arrays in the $_FILES array

index	Description
‘name’	The original name of the uploading file.
‘size’	The size of the uploaded file in bytes
‘tmp_name’	The temporary name of the uploaded file on the web server.
‘type’	The MIME type of the uploaded file as sent by the user browser.
‘error’	The error code associated with the file . Common value are UPLOAD_ERR_OK(no error),UPLOAD_ERR_INI_SIZE(file was too large), and UPLOAD_ERR_PARTIAL(upload was not completed).

PHP for working with an uploaded file

```
<?php
$tmp_name=$_FILES["file1"]["tmp_name"];
$path =getcwd() ;
$name=$path.DIRECTORY_SEPARATOR
        .$_FILES["File1"]["name"];
$success=move_uploaded_file($tmp_name,$name);
if($success)
{
$uploaded_message=$name."has been uploaded.";
Echo “$ uploaded_message”;
}
?>
```

Uploading image file

- **To upload a file an HTML for we can code**

```
<form action=“imageupload.php” method=“post”
   enctype=“multipart/form-date”>
<input type =“file” name=“file1”><br/>
<input type=“submit” value=“upload”>
</form>
```

Image.php

```
<?php
$tmp_name=$_FILES["file1"]["tmp_name"];
$path =getcwd()
$name=$path.DIRECTORY_SEPARATOR
          .$_FILES["File1"]["name"];
$success=move_uploaded_file($tmp_name,$name);
if($success)
{
$uploaded_message=$name."has been uploaded.";
echo “$uploaded_message”;
}
?>
```

Upload image with image information like width ,height and type

```
<?php
$tmp_name=$_FILES["file1"]["tmp_name"];
$path =getcwd();
echo $path;
$name=$path.DIRECTORY_SEPARATOR .
   $_FILES["file1"]["name"];
$success=move_uploaded_file($tmp_name,$name);
if($success)
{
$imageinfo = getimagesize($name);
echo $imageinfo;
echo"<br>";
$image_width= $imageinfo[0];
echo "$image_width";
echo"<br>";
$image_height=$imageinfo[1];
echo "$image_height";
echo"<br>";
$image_type=$imageinfo[2];
echo "$image_type";
echo"<br>";
$uploaded_message=$name."has been uploaded.";

echo $uploaded_message;
}
?>
```