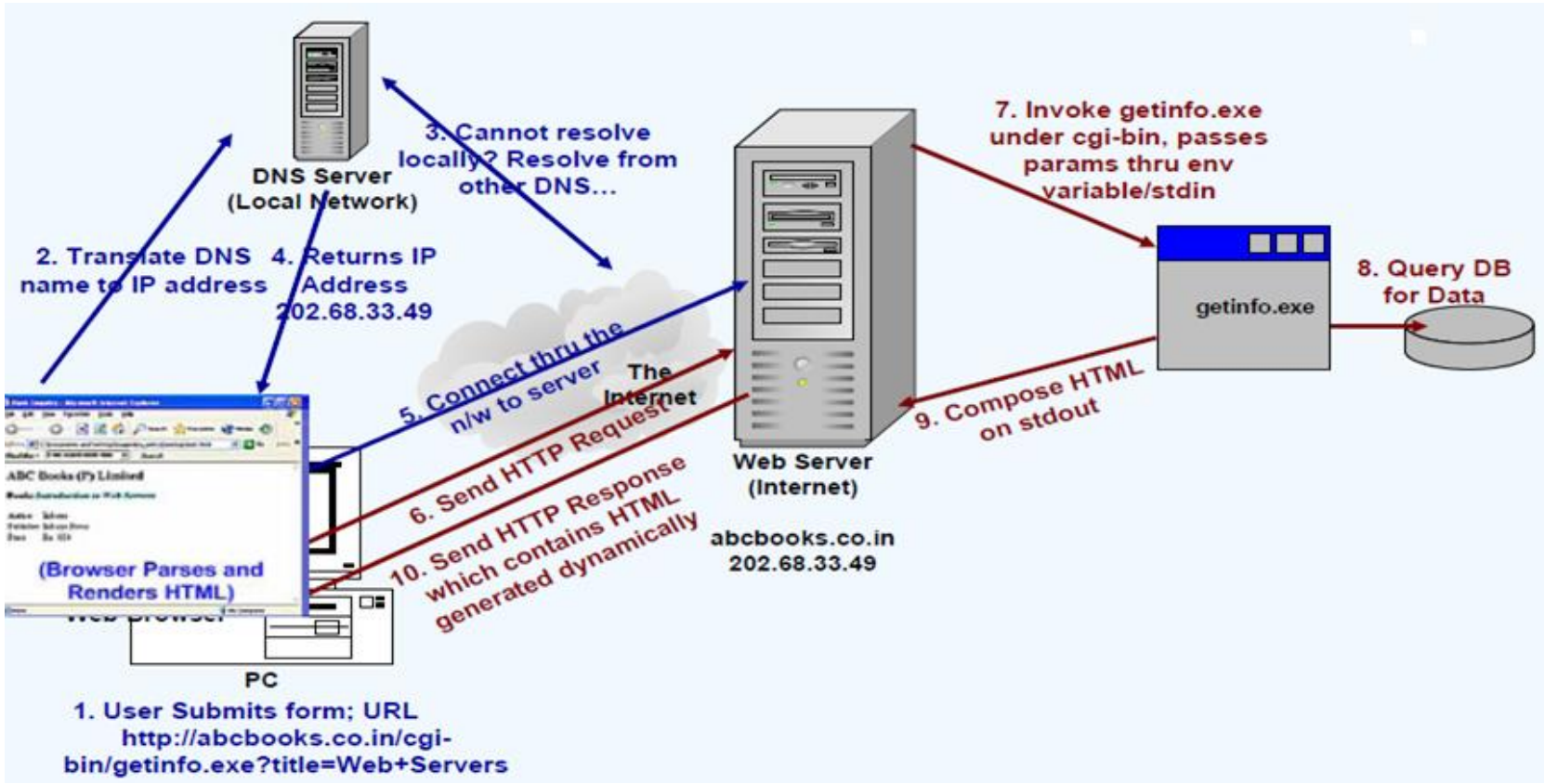
Introduction to PHP

Dynamic Server-side web application



XAMPP

- **XAMPP**
  - is a free and open source cross-platform web server solution stack package
  - an acronym for X (any of four different operating systems)
    - Microsoft Windows, Linux, Sun Solaris and Mac OS X
    - Apache, MySQL, PHP and Perl.
  - released under the GNU General Public License
  - Cross-platform (Linux, Windows, Solaris,Mac OS X)
  - Consisting of
    - the Apache HTTP Server
    - MySQL database
    - interpreters for scripts written in PHP and Perl
- Useful links
  - https://www.apachefriends.org/index.html
  - http://www.wikihow.com/Set-up-a-Personal-Web-Server-with-XAMPP

Apache Tomcat - introduction

- world's most widely used web server software
- Support Perl, Python, Tcl, and PHP
- Secure Sockets Layer and Transport Layer Security support
- Virtual hosting allows one Apache installation to serve many different Web sites
- Useful links
  - https://www.apachefriends.org/index.html
  - http://www.wikihow.com/Set-up-a-Personal-Web-Server-with-XAMPP

PHP Basics - Introduction

- PHP
  - Is a development from Personal Home Page Tools started by Rasmus Lerdorf in 1994
  - Mixture of Perl, C and Java
  - Apache module which integrates PHP into Web Server is the most popular
  - Is a web specific tool
  - Free software & has been ported to many OS
  - Supports standard network protocol – IMAP, NNTP, SMTP, POP3 & HTTP
  - Can work with wide range of Database Systems
  - Ability to process XML data and RSS feeds

PHP Basics – What Can PHP Do?

- PHP can
  - generate dynamic page content
  - create, open, read, write, delete, and close files on the server
  - collect form data
  - send and receive cookies
  - Create and manage user session
  - add, delete, modify data in your database
  - restrict users to access some pages on your website
  - encrypt data

PHP Basics - Echo & Print

- Comments represented as /* ..*/ , single line - //
- Variable names are case-sensitive
- echo - can output one or more strings
- print - can only output one string, and returns 1 always
- Example :
  - echo "This", " string", " was", " made", " with multiple parameters.";
  - print "<h2>PHP is fun!</h2>";

```
<!DOCTYPE html>
<html>
<body>

<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt1 = "PHP example";
$txt2 = “MIT hello!";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>

</body>
</html>
```

PHP Basics - PHP Variables

- Loosely Type Language
- PHP has three different variable scopes:
  - local
  - global
  - static
- Rules for PHP variables:
  - A variable starts with the $ sign, followed by the name of the variable
  - A variable name must start with a letter or the underscore character
  - A variable name cannot start with a number
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Variable names are case sensitive ($y and $Y are two different variables)

Variable scope

- A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function
- A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function
- The **global** keyword is used to access a global variable from within a function
- PHP also stores all global variables in an array called **$GLOBALS[index].**
  - The index holds the name of the variable

Example

```
<!DOCTYPE html>
<html>
<body>

        <?php
        $fname = "ABC<br/>";
        $lname = "XYZ<br/>";

        echo $fname , $lname;
        //print $fname , $lname;
        $rvalue=print $fname;
        echo $rvalue;
        ?>
</body>
</html>
```

Example

```
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
<?php
$x = 6;
$w = 15;
function myTest() {
        global $x, $w;
        $y = 10;
        $z= $x + $y + $w;
        echo "<p>Addition of three no: <br/> $x + $y + $w = $z</p>";
        $x=20;
}
myTest();
echo "<p>Variable x outside function is: $x</p>";
function myTest1() {
        $GLOBALS['w'] = $GLOBALS['x'] + $GLOBALS['w'];
        }
        myTest1();
        echo "<p>Addition of two number is: $w</p>";

        function myTest2() {
                static $m = 0;
                echo "$m <br/>";
                $m++;
        }
        myTest2();
        myTest2();
        myTest2();
        myTest2();
?>
```

PHP operators

- Arithmetic operators: +, -, *, /, %, **
- Assignment operators: =, +=, -=, /=, %=
- Comparison operators: ==, !=, ===, !==, >, <, >=, <=
- Increment/Decrement operators: ++$x, $x++, --$x, $x--
- Logical operators: and, or, xor, &&, ||, !
- String operators: ., .=

Conditional statements

- **if statement** - executes some code if one condition is true
- **if...else statement** - executes some code if a condition is true and another code if that condition is false
- **if...elseif....else statement** - executes different codes for more than two conditions
- **switch statement** - selects one of many blocks of code to be executed

Loops

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

PHP Basics – Arrays

1. **Indexed arrays** - Arrays with numeric index

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

PHP Basics – Arrays

Associative arrays - Arrays with named keys

- To create an associative array:
  - $age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
  - $age['Peter']="35"; $age['Ben']="37"; $age['Joe']="43";
- **Example : To display contents of array**
  - <?php
    $age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
    foreach($age as $x=>$x_value) {
    echo "Key=" . $x . ", Value=" . $x_value; echo "
";
    } ?>
  - <?php
    $a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
    print_r ($a);
    ?>

3. **Multidimensional arrays** - Arrays containing one or more arrays

```
$students = array(
   "Roll no.: 1" => array(
      "name" => "Sunitha",
      "email" => "Sunitha@mail.com",
   ),
   "Roll no.: 2" => array(
      "name" => "Clark",
      "email" => "clark@mail.com",
   ),
   "Roll no.: 3" => array(
      "name" => "Harish",
      "email" => "harish@mail.com",
   )
);
```

Array object methods

- array_push($arr, $val)- pushes value to a array
- array_pop($arr)- removes an element from the end of an array and returns its value
- array_reverse($arr)- reverses the order of the elements in an array.
- array_flip($arr) - interchanges the keys and the values of an Associative array
- array_unique($arr) - remove all duplicate entries in the array.
- array_keys($arr)- recover the keys from an associative array.
- array_values($arr)- receives a PHP array.
- sort($arr) - Sorts scalar array in ascending order.
- rsort($arr) - Sorts scalar array in reverse order.
- asort($arr) - Sorts associative array by values.
- arsort($arr) - Sorts associative array by values in reverse order.
- ksort($arr) - Sorts associative array by ‘Keys’.
- krsort($arr) - Sorts associative array by ‘Keys’ in reverse order

PHP Super globals

- Several predefined variables in PHP are "superglobals“
- Are always accessible
- The PHP superglobal variables are:
  - $GLOBALS – access to all global variables
  - $_SERVER - holds information about headers, paths, and script locations.
  - $_REQUEST - used to collect data after submitting an HTML form
  - $_POST - used to collect form data after submitting an HTML form with method="post"
  - $_GET - used to collect form data after submitting an HTML form with method="get"
  - $_FILES - array you can upload files from a client computer to the remote server
  - $_ENV - array containing environment variables
  - $_COOKIE – array containing all cookies
  - $_SESSION – array containing session variables.

PHP Server super global

$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server

$_REQUEST

```
<html>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // collect value of input field
   $name = $_REQUEST['fname'];
   if (empty($name)) {
                echo "Name is empty";
   } else {
                echo $name; } }
?>
</body>
</html>
```

$_GET

```
<html> <body>
<form method="get" action="gettdest.php">
  Name: <input type="text" name="fname">
  <input type="submit">
</form> </body> </html>
```

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "GET") {
   // collect value of input field
   $name = $_GET['fname'];
   if (empty($name)) {
      echo "Name is empty";
   } else {
      echo $name;
   }
}
?>
```

$_POST

```
<html>
<body>

<form method="post" action="postdest.php">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

</body>
</html>
```

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // collect value of input field
   $name = $_POST['fname'];
   if (empty($name)) {
      echo "Name is empty";
   } else {
      echo $name;
   }
}
?>
```

Connecting to MySQL through PHP

- Object oriented mysqli
  - <?php
    $servername = "localhost"; $username = "username";
    $password = "password"; $database = “eventdb”;
    // Create connection
    $conn = new mysqli($servername, $username, $password, $database);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    echo "Connected successfully";
    ?>
- Procedural mysqli
  - <?php
    $servername = "localhost";
    $username = "username";
    $password = "password"; $database = “eventdb   // Create connection
    $conn = mysqli_connect($servername, $username, $password, $database);   // Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }
    echo "Connected successfully";
    ?>

PHP and MySQL

Connecting to MySQL through PHP

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error); }
// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
   echo "Database created successfully";
} else { echo "Error creating database: " . $conn->error; }
$conn->close();
?>
```

Inserting record

```
<?php
$servername = "localhost";
$username = "a";
$password = "";
$dbname = "mydb";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error); }
$sql = "INSERT INTO dept (name) VALUES ('John')";
if ($conn->query($sql) === TRUE) {
   echo "New record created successfully";
} else {  echo "Error: " . $sql . "<br>" . $conn->error; }
$conn->close();
?>
```

Prepared statement

- Preparing statements and binding parameters
  - SQL statement template is prepared with unspecified parameters and sent to database
  - DBMS parses and compiles the statement and stores the result
  - The parameter value is bound to SQL statement and then executed
- **Advantage**
  - It reduces parsing time as preparation for query is done only once
  - It minimizes bandwidth to the server
  - Useful against SQL injection as parameter values are communicated later using different protocol

Using prepared statement

```
<?php
$servername = "localhost"; $username = “root"; $password = ""; $dbname = "mydb";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {  die("Connection failed: " . $conn->connect_error); }

$sql = $conn->prepare("INSERT INTO dept (name) VALUES (?)");
$sql->bind_param("s",$dname);
//set parameter variable types string(s), integer(i), double(d) and blob (b)
$dname="Rahul";
$sql->execute();

$dname="Mary";
$sql->execute();
   echo "New record created successfully";
$sql->close();
$conn->close();
?>
```

Multiquery

```
<?php
$servername = "localhost"; $username = "a"; $password = ""; $dbname = "mydb";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {  die("Connection failed: " . $conn->connect_error);  }
$sql = "INSERT INTO dept (name) VALUES ('ABC');";
$sql .= "INSERT INTO dept (name) VALUES ('XYZ');";
$sql .= "INSERT INTO dept (name) VALUES ('LMN')";

if ($conn->multi_query($sql) === TRUE) {
   echo "New record created successfully"; } else {
   echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

Selecting records

```
<?php
$servername = "localhost"; $username = “root"; $password = ""; $dbname = "mydb";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error); }
$sql = "SELECT id, name FROM dept";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
   // output data of each row
   while($row = $result->fetch_assoc()) {
      echo "id: ".$row["id"]." - Name: ".$row["name"]."<br>"; }
} else { echo "0 results"; } $conn->close();
 ?>
```

Include & require statement

- takes all the text/code/markup that exists in the specified file and copies it into the file that uses the statement
- **The include and require statements 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
- Syntax
  - include '*filename*'; or require '*filename*';
- Standard menu
  - <?php
    echo 'Home -
    HTML Tutorial -
    CSS Tutorial -
    JavaScript Tutorial -
    PHP Tutorial';
    ?>
- Can be included
  -
    <?php include 'menu.php';?>

The header function

- sends a raw HTTP header to a client
- must be called before any actual output is sent
- Syntax : header(string,replace,http_response_code)
  - <?php // Date in the past
    header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
    header("Cache-Control: no-cache");
    ?>
- To redirect to another page
  - header('Location: index.php ');

Cookie Handling using PHP

- Create a cookie with the setcookie() command
- retrieve it with the $_COOKIE superglobal
- The setcookie() function must be before the <html> tag.
  - setcookie(name, value, expire, path, domain);
  - setcookie(“username”, “jack”, time()+(60*60*24));
- To display cookie contents
  - <p>You entered <?php echo $_COOKIE[“username”] ?> as the User Name
- To delete cookie contents
  - setcookie(“username“, “”, time()-3600);

Session Handling using PHP
Destroying session variables

- The unset() function is used to free the specified session variable:
  - <?php
    session_start();
    if(isset($_SESSION['views']))
      unset($_SESSION['views']);
    ?>
- To unset session variables
  - session_unset();
- You can also completely destroy the session by calling the session_destroy() function:
  - <?php
    session_destroy();
    ?>

Example

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

<?php
$_SESSION["color"] = "yellow";

print_r($_SESSION);
unset($_SESSION["color"]);
if(!isset($SESSION["color"]))
                        echo "Session variable not defined";
else
                        echo $_SESSION["color"];
?>

</body>
</html>
```

$_COOKIE

```
<?php
$cookie_name = "user";
$cookie_value = 2;
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
   setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day

} else {
$newcookie_value=$_COOKIE[$cookie_name]+1;
setcookie($cookie_name, $newcookie_value, time() + (86400 * 30), "/");
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

$_SESSION

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

Difference between cookies & sessions

- Cookies are returned and stored in the user's browser, session data is stored on the web server.
- The life span of a cookie can be set to almost any duration of your choosing. PHP sessions have a predetermined short life. The exact life span depends on how your web host has configured PHP on your server.
- Depending on how your web server is configured, session data is often stored in a public temporary directory on the server.
- **When to use sessions rather than cookies**
  - When you need the data stored on the server and not your user's browser
  - When the data is transient, and only relevant for the current browsing session

AJAX

- AJAX = Asynchronous JavaScript and XML.
- AJAX is a technique for creating fast and dynamic web pages
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server
- It is possible to update parts of a web page, without reloading the whole page.
- AJAX is based on internet standards, and uses a combination of:
  - XMLHttpRequest object (to retrieve data from a web server)
  - JavaScript/DOM (to display/use the data

- The XMLHttpRequest object is used to exchange data with a server behind the scenes
- *variable* = new XMLHttpRequest();
- Old versions of Internet Explorer (IE5 and IE6) use an ActiveX Object:
- *variable* = new ActiveXObject("Microsoft.XMLHTTP");
- var xhttp;
  if (window.XMLHttpRequest) {
  xhttp = new XMLHttpRequest();
  } else {
  // code for IE6, IE5
  xhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }

- we use the open() and send() methods of the XMLHttpRequest object:
- xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
- open(method, url, async) Specifies the type of request
- method: the type of request: GET or POST
- url: the server (file) location
- async: true (asynchronous) or false (synchronous)
- send() Sends the request to the server (used for GET)
- send(string)Sends the request to the server (used for POST)

- xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", tru e);
  xhttp.send();
- xhttp.open("POST", "ajax_test.asp", true);
  xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhttp.send("fname=Henry&lname=Ford");

onreadystatechange event

- The onreadystatechange event is triggered every time the readyState changes.
- The readyState property holds the status of the XMLHttpRequest.
- readyState: Holds the status of the XMLHttpRequest. Changes from 0 to 4:
  - 0: request not initialized
  - 1: server connection established
  - 2: request received
  - 3: processing request
  - 4: request finished and response is ready
- status 200: "OK"

  404: Page not found

File Upload using PHP

- 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
  - $_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.

$_FILES

Uploading to temporary

//HTML

```
<html>
  <body>

    <form action = "fileupload.php" method = "POST" enctype = "multipart/form-data">
      <input type = "file" name = "image" />
      <input type = "submit"/>


    </form>

  </body>
</html>
```

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

```
<?php
  if(isset($_FILES['image'])){
    $errors= array();
    $file_name = $_FILES['image']['name'];  $file_size = $_FILES['image']['size'];  $file_tmp =
$_FILES['image']['tmp_name'];   $file_type = $_FILES['image']['type'];
                $temp=explode('.',$_FILES['image']['name']);
                    $file_ext=end($temp);
                    $expensions= array("jpeg","jpg","png");
    if(in_array($file_ext,$expensions)=== false){
      $errors[]="extension not allowed, please choose a JPEG or PNG file."; }
    if($file_size > 2097152) {
      $errors[]='File size must be 2 MB or less'; }
    if(empty($errors)==true) {
      move_uploaded_file($file_tmp,"images/".$file_name);
      echo "Success"; echo "<ul>
        <li>Sent file: ".$_FILES['image']['name']."
                                        <li>File size: ".$_FILES['image']['size']."
        <li>File type:".$_FILES['image']['type']."
      </ul>"; }else{  print_r($errors);} }
```

PHP Basics - preg_match() function

- Perform a regular expression match
- int **preg_match** (string $pattern , string $subject [, array &$matches [, int $flags = 0 [, int $offset = 0]]])
  - pattern to search for, as a string.
  - subject is the input string.
  - matches is filled with the results of search.
  - If this flag is passed, for every occurring match the string offset will also be returned.
  - The optional parameter offset can be used to specify the alternate place from which to start the search
- <?php
  // The "i" after the pattern delimiter indicates a case-insensitive search
  if (preg_match("/php/i", "PHP is the web scripting language of choice.")) {
  echo "A match was found.";
  } else {
  echo "A match was not found.";
  }
  ?>
- preg_match("/^[a-zA-Z]*$/",$name)
- preg_match('/^\d{10}**$**/', $mobile)
- preg_match("/^[a-zA-Z0-9_]+\@[a-zA-Z0-9]+\.[a-zA-Z0-9]/",$email)

Data validation using PHP

- Name validation

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

Data validation using PHP

- **Form – usage of input of name**

  Name:

  ```
  <input type="text" name="name" value="<?php echo $name;?>">
    <span class="error">* <?php echo $nameErr;?></span>
     <br><br>
  ```

- **Mirroring of content after form**
  - ```
    <?php
    echo "<h2>Your Input:</h2>";
    echo $name;
    echo "<br>";
    …….
    ```
  - **?>**
    ```

JSON

- The JSON extension implements the **JavaScript Object Notation** data-interchange format.
- From PHP 5.2.0, the JSON functions are enabled by default. There is no inst

Function	Description
json_decode()	Decodes a JSON string
json_encode()	Encode a value to JSON format
json_last_error()	Returns the last error occurred
json_last_error_msg()	Returns the error string of the last json_encode() or json_decode() call

- The json_decode() function is used to decode or convert a JSON object to a PHP object.
- json_decode(*string, assoc, depth, options*)

Parameter	Description
string	Required. Specifies the value to be decoded
assoc	Optional. Specifies a Boolean value. When set to true, the returned object will be converted into an associative array. When set to false, it returns an object. False is default
depth	Optional. Specifies the recursion depth. Default recursion depth is 512
options	Optional. Specifies a bitmask (JSON_BIGINT_AS_STRING, JSON_INVALID_UTF8_IGNORE, JSON_INVALID_UTF8_SUBSTITUTE, JSON_OBJECT_AS_ARRAY, JSON_THROW_ON_ERROR)

- ```
  <?php
  $jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

  $obj = json_decode($jsonobj);

  echo $obj->Peter;
  echo $obj->Ben;
  echo $obj->Joe;
  ?>
  ```
- ```
  <?php
  $jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

  $arr = json_decode($jsonobj, true);

  echo $arr["Peter"];
  echo $arr["Ben"];
  echo $arr["Joe"];
  ?>
  ```

- The json_encode() function is used to encode a value to JSON format.
- json_encode(*value, options, depth)*

Parameter	Description
value	Required. Specifies the value to be encoded
options	Optional. Specifies a bitmask (JSON_FORCE_OBJECT, JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS, JSON_INVALID_UTF8_IGNORE, JSON_INVALID_UTF8_SUBSTITUTE, JSON_NUMERIC_CHECK, JSON_PARTIAL_OUTPUT_ON_ERROR, JSON_PRESERVE_ZERO_FRACTION, JSON_PRETTY_PRINT, JSON_UNESCAPED_LINE_TERMINATORS, JSON_UNESCAPED_SLASHES, JSON_UNESCAPED_UNICODE, JSON_THROW_ON_ERROR)
depth	Optional. Specifies the maximum depth

- ```
  <?php
  $age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);

  echo json_encode($age);
  ?>
  ```
  {"Peter":35,"Ben":37,"Joe":43}
- ```
  <?php
  $cars = array("Volvo", "BMW", "Toyota");

  echo json_encode($cars);
  ?>
  ```
- **["Volvo","BMW","Toyota"]**

```
<!DOCTYPE html>
<html>
<body>
<?php
// An invalid json string
$string = "{'Peter':35,'Ben':37,'Joe':43}";

echo "Decoding: " . $string;
json_decode($string);
echo "<br>Error: ";

switch (json_last_error()) {
  case JSON_ERROR_NONE:
    echo "No errors";
    break;
  case JSON_ERROR_DEPTH:
    echo "Maximum stack depth exceeded";
    break;
  case JSON_ERROR_STATE_MISMATCH:
    echo "Invalid or malformed JSON";
    break;
  case JSON_ERROR_CTRL_CHAR:
    echo "Control character error";
    break;
  case JSON_ERROR_SYNTAX:
    echo "Syntax error";
    break;
  case JSON_ERROR_UTF8:
    echo "Malformed UTF-8 characters";
    break;
  default:
    echo "Unknown error";
    break;
}
?>
</body>
</html>
```

StoredProcedure

- A stored procedure is a set of SQL commands that have been compiled and stored on the database server.
- Once the stored procedure has been “stored”, client applications can execute the stored procedure over and over again without sending it to the database server again and without compiling it again.
- Stored procedures improve performance by reducing network traffic and CPU load.

Example

```
DELIMITER $$

CREATE PROCEDURE GetCustomerdata()
BEGIN
    SELECT *
    FROM customers;
END$$
```

```
<?php
 $host = 'localhost';
   $dbname = 'mydb';
   $username = 'root';
   $password = '';
      try {
         $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
         // execute the stored procedure
         $sql = 'CALL GetCustomerdata()';
         // call the stored procedure
         $q = $pdo->query($sql);
         $q->setFetchMode(PDO::FETCH_ASSOC);
      } catch (PDOException $e) {
         die("Error occurred:" . $e->getMessage());
      }
      ?>
      <table>
         <tr>
            <th>Customer Name</th>
            <th>Address</th>
         </tr>
         <?php while ($r = $q->fetch()): ?>
            <tr>
               <td><?php echo $r['name'] ?></td>
               <td><?php echo $r['address'] ?></td>            </tr>
         <?php endwhile; ?>
      </table>
```