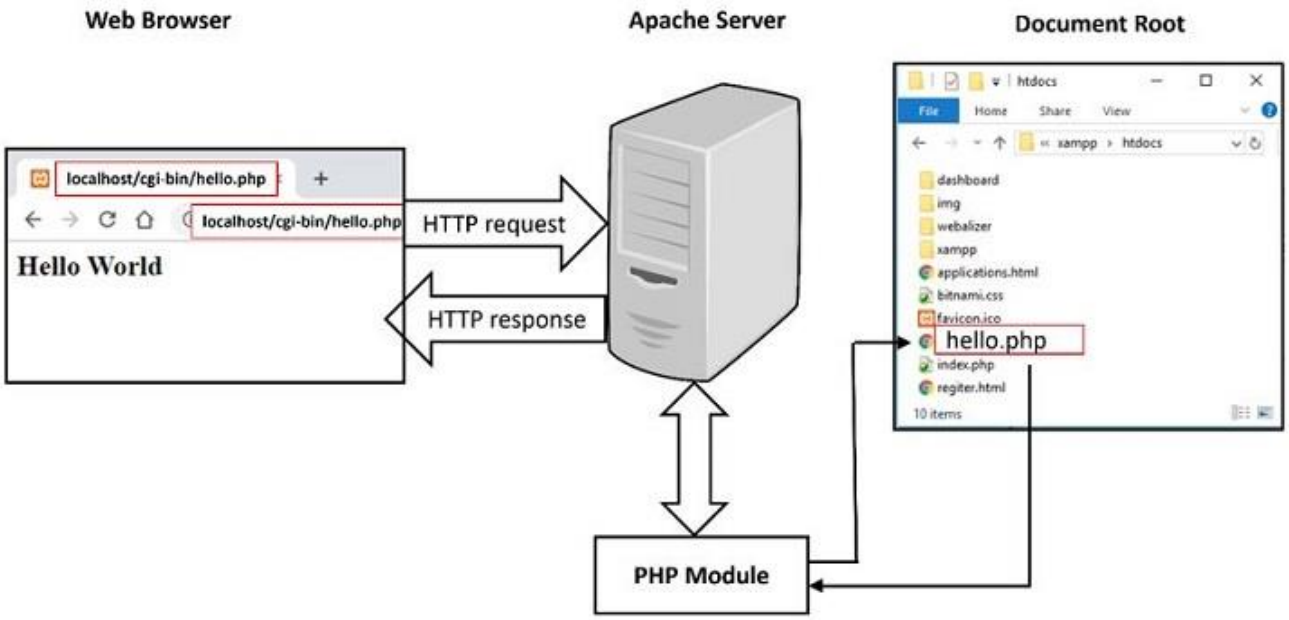
Form Handling: Creating HTML Form, Handling HTML Form data in PHP. File Inclusion (Include ( ), Require ( )).
Session Handling: Definition, session_start (), session_id (), session_destroy () session variables.

PHP - Web Concepts

PHP is a server-side scripting language that is used to create dynamic webpages. It is one of the most popular programming languages for web development. This chapter aims to let you get familiarized with certain important concepts of web application development using PHP.

A web-based application is a collection of webpages. A webpage is mainly created with HTML tags. HTML consists of different HTML tags which are required to define the appearance of page elements like text, image, table, etc. Hence, HTML essentially creates a static webpage.

A Web application is hosted on a HTTP server with PHP module installed. The browser acts as a http client, to establish communication with the server, following HTTP protocol.



How to Add Dynamic Content on a Webpage?

To add dynamic content io a webpage, there are two possibilities.

JavaScript is a client-side scripting language, that can access the HTML document object model and render dynamic content on the client browser. JavaScript code can be embedded in HTML page.

The browser may collect data from the user in the form of HTML form elements and send it to a HTTP server for processing. PHP is a widely used Server-side processing language. PHP script can also be embedded inside HTML page.

Example

In the following script, JavaScript code embedded in HTML renders the current date as per the client browser, and the PHP code displays the current date as per the server, where this script is hosted.

```
<!DOCTYPE html>
<html>
<body>
  <script type="text/JavaScript">
    document.write("Client's date :"+Date()+"\n");
  </script>
  <?php
    date_default_timezone_set("Asia/Calcutta");
    echo "server's date is " . date("Y-m-d") . "\n";
    echo "The time is " . date("h:i:sa");
  ?>
</body>
</html>
```

PHP can intercept and process the data from HTML forms. This allows you to collect information from your users. The next chapter discusses PHPâ□□s form handling.

PHP can be used to interact with databases such as MySQL and PostgreSQL. This allows you to store and retrieve data from your database, and dynamically populate the web pages or to power the web applications. PHP includes mysql, mysqli and PDO extensions for database handling.

PHP can handle the data received from the client with HTTP GET as well as POST methods. We shall discuss in detail, how PHP handles GET/POST methods in one of the latter chapters.

HTTP is a stateless protocol. However, it allows Sessions and cookies to be maintained on server and client respectively. PHP can be used to create and manage sessions and cookies. Sessions allow you to track individual users as they navigate your website, while cookies allow you to store information on the user's computer for later use. In of the subsequent chapters, we shall learn how PHP handles sessions and cookies.

PHP can be used to upload files to your web server. This allows you to create web applications that allow users to upload files, such as images, videos, or documents.

You can use PHP to create a login page for your website. When the user enters their username and password, PHP can check the database to see if the user is valid. If the user is valid, PHP can log the user in and redirect them to the main page of your website.

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.

Display Images Randomly

The PHP **rand()** function is used to generate a random number.i This function can generate numbers with-in a given range. The random number generator should be seeded to prevent a regular pattern of numbers being generated. This is achieved using the **srand()** function that specifies the seed number as its argument.

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.

Example

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_POST["name"] || $_POST["age"] ) {
     if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
        die ("invalid name and name should be alpha");
     }

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

     exit();
```

```
  }
?>
<form action = "<?php <b>$_PHP_SELF</b> ?>" method = "POST">
  Name: <input type = "text" name = "name" />
  Age: <input type = "text" name = "age" />
  <input type = "submit" />
</form>
```

It will produce the following result −

- The PHP default variable **$_PHP_SELF** is used for the PHP script name and when you click "submit" button then same PHP script will be called and will produce following result −
- The method = "POST" is used to post user data to the server script. There are two methods of posting data to the server script which are discussed in PHP GET & POST chapter.

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.

PHP - Form Handling

HTML Forms play an important role in PHP web applications. Although a webpage composed purely with HTML is a static webpage, the HTML form component is an important feature that helps in bringing interactivity and rendering dynamic content. PHP's form handling functionality can validate data collected from the user, before processing.

What is Form Handling ?

An HTML Form is a collection various form controls such as text fields, checkboxes, radio buttons, etc., with which the user can interact, enter or choose certain data that may be either locally processed by JavaScript (client-side processing), or sent to a remote server for processing with the help of server-side programming scripts such as PHP.

For example - suppose a user enters their name and email, clicks submit and you can use PHP to handle this data.

HTML Form Structure

One or more form control elements are put inside <form> and </form> tags. The **form element** is characterized by different attributes such as name, action, and method.

<form [attributes]>

Form controls

</form>

Form Attributes

Out of the many attributes of the HTML form element, the following attributes are often required and defined −

Action Attribute

a string representing the URL that processes the form submission. For example, **http://example.com/test.php**. To submit the for-data to the same PHP script in which the HTML form is defined, use the PHP_SELF server variable −

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

Enctype Attribute

specifies the method using which the form-data should be encoded before sending it to the server. Possible values are −

- application/x-www-form-urlencoded − The default value.
- multipart/form-data − Use this if the form contains <input> elements with type=file.
- text/plain − Useful for debugging purposes.

Method Attribute

a string representing the HTTP method to submit the form with. The following methods are the possible values of method attribute −

- **post** − The POST method; form data sent as the request body.
- **get (default)** − The GET; form data appended to the action URL with a "?" separator. Use this method when the form has no side effects.
- **dialog** − When the form is inside a <dialog>, closes the dialog and causes a submit event to be fired on submission, without submitting data or clearing the form.
-

Name Attribute

The name of the form. The value must not be the empty string, and must be unique if there are multiple forms in the same HTML document.

Target Attribute

a string that indicates where to display the response after submitting the form. Should be one of the following −

- **_self (default)** − Load into the same browsing context as the current one.
- **_blank** − Load into a new unnamed browsing context.
- **_parent** − Load into the parent browsing context of the current one.
- **_top** − Load into the top-level browsing context (an ancestor of the current one and has no parent).

Hence, a typical HTML form, used in a PHP web application looks like –

```
<form name="form1" action="<?php echo $_SERVER['PHP_SELF'];?>" action="POST">
  Form controls
</form>
```

Form Elements

A HTML form is designed with different types of controls or elements. The user can interact with these controls to enter data or choose from the available options presented. Some of the elements are described below −

Input Element

The input element represents a data field, which enables the user to enter and/or edit the data.

The type attribute of INPUT element controls the data. The INPUT element may be of the following types −

Text

A text field to enter a single line text.

```
<input type="text" name="employee">
```

Password

A single line text filed that masks the entered characters.

```
<input type="password" name="pwd"><br>
```

Checkbox

A rectangular checkable box which is a set of zero or more values from a predefined list.

<input type="checkbox" id="s1" name="sport1" value="Cricket">

<label for="s1">I like Cricket</label>

<input type="checkbox" id="s2" name="sport2" value="Football">

<label for="s2">I like Football</label>

<input type="checkbox" id="s3" name="sport3" value="Tennis">

<label for="s3">I like Tennis</label>

Radio

This type renders a round clickable button with two states (ON or OFF), usually a part of multiple buttons in a radio group.

<input type="radio" id="g1" name="gender" value="Male">

<label for="g1">Male</label>

<input type="radio" id="g2" name="female" value="Female">

<label for="g2">Female</label>

File

The input type renders a button captioned file and allows the user to select a file from the client filesystem, usually to be uploaded on the server. The form's enctype attribute must be set to "multipart/form-data"

<input type="file" name="file">

Email

A single line text field, customized to accept a string conforming to valid email ID.

URL

A single line text filed customized to accept a string conforming to valid URL.

Submit

This input element renders a button, which when clicked, initiates the the submission of form data to the URL specified in the action attribute of the current form.

<input type="submit" name="Submit">

Select Element

The select element represents a control for selecting amongst a set of options. Each choice is defined with option attribute of Select Control. For example –

```
<select name="Subjects" id="subject">
  <option value="Physics">Physics</option>
  <option value="Chemistry">Chemistry</option>
  <option value="Maths">Maths</option>
  <option value="English">English</option>
</select>
```

Form Example

Let us use these form elements to design a HTML form and send it to a PHP_SELF script

When you fill out a form and hit the 'Submit' button, PHP steps in to get the information you have given. It uses something called $_POST to get the details from fields like your name, email, website, comment and gender. To keep things safe and secure, PHP employs the htmlspecialchars() function, which helps to block any potentially harmful code. After processing everything, PHP then shows you the information you've entered right on the webpage.

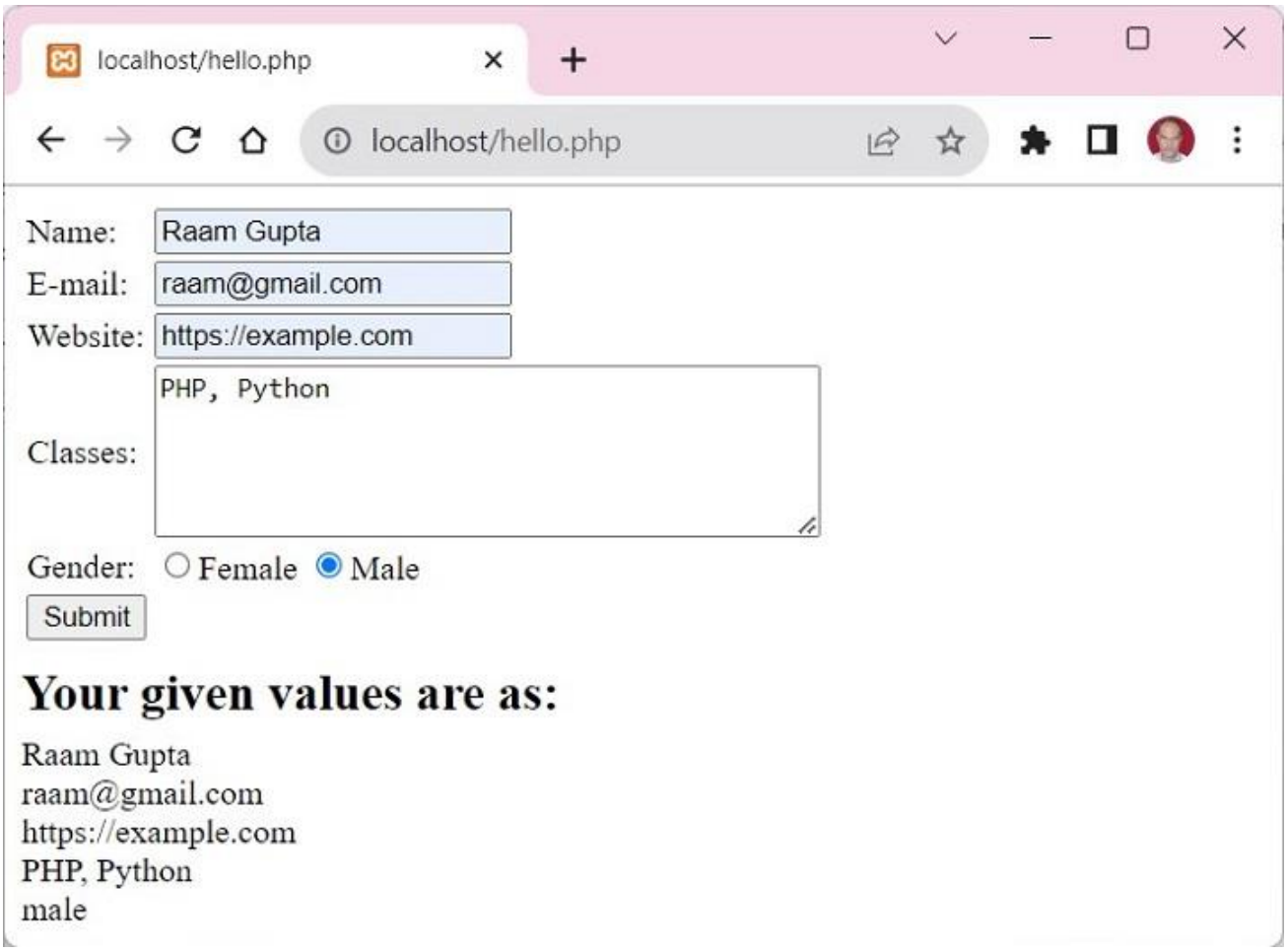
```
<html>
<body>
  <form method = "post" action = "<?php
    echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
    <table>
      <tr>
        <td>Name:</td>
        <td><input type = "text" name = "name"></td>
      </tr>
      <tr>
        <td>E-mail: </td>
        <td><input type = "email" name = "email"></td>
      </tr>
```

```
    <tr>
      <td>Website:</td>
      <td><input type = "url" name = "website"></td>
    </tr>
    <tr>
      <td>Classes:</td>
      <td><textarea name = "comment" rows = "5" cols = "40"></textarea></td>
    </tr>
    <tr>
      <td>Gender:</td>
      <td>
        <input type = "radio" name = "gender" value = "female">Female
        <input type = "radio" name = "gender" value = "male">Male
      </td>
    </tr>
    <td>
      <input type = "submit" name = "submit" value = "Submit">
    </td>
  </table>
</form>
<?php
  $name = $email = $gender = $comment = $site = "";

  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST["name"];
    $email = $_POST["email"];
    $name = $_POST["name"];
    $comment = $_POST["comment"];
```

```
      $gender = $_POST["gender"];
      $site = $_POST["website"];
   }
   echo "<h2>Your given values are as:</h2>";
   echo $name;
   echo "<br>";

   echo $email;
   echo "<br>";

   echo $site;
   echo "<br>";

   echo $comment;
   echo "<br>";

   echo $gender;
 ?>
</body>
</html>
```

Output

It will produce the following output –



PHP - Form Validation

The term "Form Validation" refers to the process of ascertaining if the data entered by the user in various form elements is acceptable for further processing. Validation of data before its subsequent processing avoids possible exceptions and runtime errors.

Types of Form Validation

Validation can be done both on the client-side and on the server-side. When the client submits the form, the form data is intercepted by the PHP script running on the server. Using various functions available in PHP, the server-side form validation can be done.

- **Client-side Validation:** This happens in the user's web browser before the form submission. It provides instant feedback to users. This is commonly done with JavaScript.
- **Server-side Validation:** After the submission of the form, server-side validation happens. It is important for security and ensures that data is checked on the server even when client-side validation is turned off.

How to Validate Forms in PHP

To validate forms in PHP you will have to follow the below steps −

- **Create a Form:** First you need to create a simple HTML form.
- **Collect Form Data:** Then use PHP to collect the data after the form is submitted.
- **Perform Validation:** After that you have to check if the data meets specific criteria.
- **Provide Feedback:** Inform the user if there are any errors or if the submission is successful.

Client-Side Validation

The new input controls as per the HTML5 specifications have in-built validation. For example an input element of the type 'email', even though is a text field, is customized to accept a string that is according to email address protocol.

Validation takes place before the data is submitted to the server. Same thing is true with other input types such as URL, number, etc.

Example

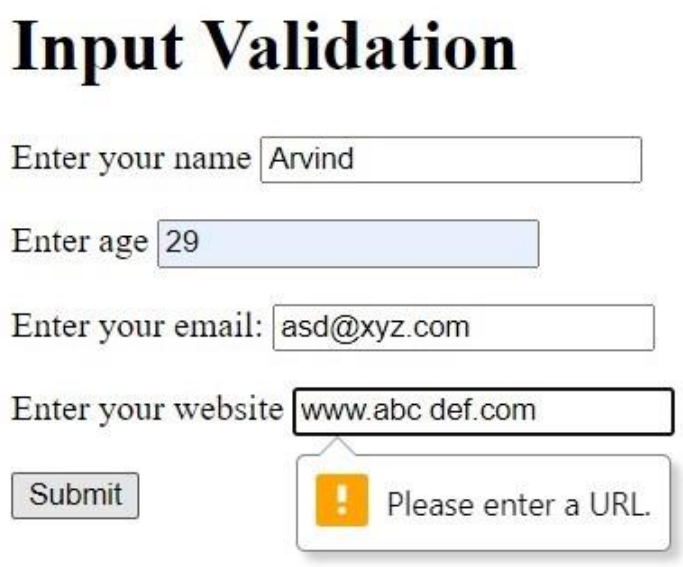
Given below is an HTML form with input elements of number type, email type and URL type. If you enter data that is not as per the required format, a suitable error message is flashed as you try to submit the form.

```
<h1>Input Validation</h1>

<form>

  <p><Label for "name">Enter your name</label>

  <input type = "text" id="name" name="name"></p>

  <p><label for="age">Enter age</label>

  <input type = "text" id = "age" name="age"></p>

  <p><label for="email">Enter your email:</label>

  <input type="text" id="email" name="email"></p>

  <p><label for="URL">Enter your website<label>

  <input type = "text" id="URL" name="url"></p>

  <input type="submit">

</form>
```

The number type text field shows up/down counter arrows on the right. Only number is accepted, and can be incremented or decremented.

If the data in email field is invalid, you get the error message flashed as below.

Similarly, any incorrect format for the URL also flashes the error as shown –



Server-Side Validation

The validation on the server-side with PHP comes into picture, either when the form data passes the client-side validation, or there's no validation on the client side at all.

In the HTML form used in the above example, let us remove all special input types and use all text fields of text type. The form is submitted with POST method to hello.php on the server.

```
<form action="hello.php" method="POST">
  <p><Label for "name">Enter your name</label>
  <input type = "text" id="name" name="name"></p>
  <p><label for="age">Enter age</label>
  <input type = "text" id = "age" name="age"></p>
  <p><label for="email">Enter your email:</label>
  <input type="text" id="email" name="email"></p>
  <p><label for="URL">Enter your website<label>
  <input type = "text" id="URL" name="url"></p>
  <input type="submit">
</form>
```

Form is Empty

If the user (may be inadvertently) clicks the submit button, you can ask PHP to display the form again. You need to check if the $_POST array has been initialized with isset() function. If not, the header() function redirects the control back to the form.

```
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (isset($_POST)) {
      header("Location: hello.html", true, 301);
      exit();
    }
    // form processing if the form is not empty
  }
?>
```

Example

You can also check if any of the fields is empty at the time of submitting the form.

```
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    foreach($_POST as $k=>$v) {
      if (empty($v)==true) {
        echo "One or more fields are empty \n";
        echo "<a href = 'hello.html'>Click here to go back </a>";
        exit;
      }
      else
      echo "$k => $v \n";
    }
  }
?>
```

Age field is non-numeric

In the HTML form the input field for name is of text type, hence it can accept any characters. However, we want it to be numeric. This can be ensured by is_numeric() function

```
<?php
```

```
  if (is_numeric($_POST["age"])==false) {
    echo "Age cannot be non-numeric \n";
    echo "<a href = 'hello.html'>Click here to go back</a>";
  }
?>
```

PHP also has is_string() function to check if a filed contains a string or not. Two other functions, trim() and htmlspecialchars() are also useful for form validation.

- **trim()** − Removes whitespace from the beginning and end of a string
- **htmlspecialchars()** − Converts special characters to HTML entities to prevent cross-site scripting (XSS) attacks.

Why is Form Validation Important?

Form validation is important for a number of reasons.

- **Security:** It helps to prevent malicious attacks like SQL injection and cross-site scripting (XSS).
- **Data integrity:** Validating data makes sure it is correct and usable.
- **User Experience:** It provides users with instant feedback when they make errors, which improves their experience.

Following are the rules for form validation −

- **Required Fields:** You need to make sure important fields are not empty.
- **Email Validation:** You can check if the email address is valid or not.
- **Number Validation:** You should allow only numbers for fields like age or phone number.
- **URL Validation:** Need to check if a valid website URL is entered by the user.
- **Length Check:** You have to limit how many characters a user can enter in the form.
- **Pattern Matching:** You can also use regular expressions to allow only specific characters.

Avoid $_SERVER["PHP_SELF"] Exploits

$_SERVER["PHP_SELF"] gets the current page's filename. This is useful, but it can be dangerous if not used correctly. Hackers can use this to inject malicious code into your website. For example, imagine your form action is this –

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

And the URL is changed to –

http://example.com/form.php/<script>alert('Hacked!')</script>

The above script can run and harm your website. So to avoid this you can use the **htmlspecialchars()** function. It turns special characters into harmless ones.

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

If someone tries to inject code, it will now be shown as plain text rather than executed as a script. This prevents harmful scripts from running.

Final Example

Here is the final example to avoid exploits. This means that even if someone attempts to insert harmful code, it will have no effect on your website.

```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post">
  <label for="name">Enter your name:</label>
  <input type="text" id="name" name="name">
  <input type="submit">
</form>
```

PHP - File Inclusion

When developing websites, we generally have to reuse the same information or code in many places. For example, we might want the same header, footer or menu across all pages. PHP allows us to use file inclusion instead of writing the same code many times!

File inclusion saves time, organizes our code and allows us to make simple changes. When we make a change to one file, that changes all other files that contain that file.

Types File Inclusion in PHP

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

- Include() Function
- Require() Function
- Include_once() and Require_once() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousand of files just change included file.

The include() Function

The include() 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 **include()** function generates a warning but the script will continue execution.

Syntax

Below is the syntax of the **include**function –

include 'filename.php';

Here, **filename.php** is the file you want to include. It can be a relative or absolute path.

Example

Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

Home

ebXML

AJAX

PERL

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

<?php include("menu.php"); ?>

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

Output

It will produce the following result –

Advantages of include() Method

Using the include() method in PHP has several advantages −

- Adding a file allows you to reuse the same content on other pages without having to change any code.

- You can add basic website elements like headers, footers, and menus to various pages.
- If the included file has an error, include() will only display a warning message and the script will continue to run.
- If you need to change the included file (for example, to update the header or footer), do so only once, and the changes will be reflected on all pages that use it.

The 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.

So 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

Below is the syntax of the **require**function –

require 'filename.php';

Here, **filename.php** is a file you want to require. It can be a relative or absolute path.

Example

You can try using above example with require() function and it will generate same result. But if you will try following two examples where file does not exist then you will get different results.

<?php include("xxmenu.php"); ?>

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

Output

This will produce the following result –

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

Now lets try same example with require() function.

<?php require("xxmenu.php"); ?>

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

his 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 your PHP Server configuration.

Advantages of require() Method

Using the require() method in PHP has many advantages −

- If the required file is important for the page (for example, a database connection file), require() makes sure the script stops and displays an error if it is not found, preventing the page from loading with limited functionality.
- The function makes sure all necessary files are always available for proper execution. This is useful for files that the website need to work properly.
- Like include(), require() allows you to reuse code across multiple pages.
- With require(), you only need to include the required code once, which reduces duplication.

Difference between include() and require() Methods

The require statement is also used to include a file within PHP code. However, there is one important difference between include and require: if a file is included using the include line but PHP cannot find it, the script will continue to run.

- The include() function shows a warning and continues execution if the file is missing. While the require() method shows a fatal error and stops execution if the file is missing.
- The include() function is basically used for non-critical files for example- header, footer. While require() is used for important files like database connection, configuration.
- The include() method can include a file multiple times unless include_once() is used. And the require() includes a file multiple times unless require_once() is used.
- The include() continues execution even if the file is not present of missing. And require() stops script execution if the file is missing.

PHP GET & POST

Since PHP is mostly used for web application development, the data sent by the browser client is mainly with the GET and POST types of HTTP request methods. The HTTP protocol also defines other methods for sending the request to the server. They are PUT, DELETE, HEAD and OPTIONS (in addition to GET and POST methods). In this chapter, we shall concentrate on how PHP handles the GET and POST methods.

The GET Method

What is the HTTP GET Method?

The GET request is generally used to get data from the server. It is also used to append the form data to the URL in the format of name-value pair. The length of the URL will stay limited if we use GET. It also helps to submit the bookmark of the result. GET is better in the case when the data do not require any type of security or have any images or files.

Features of GET

Below are some of the features of the GET request:

- It becomes very easy to bookmark the data sent to the server.
- The length restriction when using the GET method is limited.
- GET request is only used to retrieve the data from the address bar of the browser.
- GET request helps us to store data easily.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_GET["name"] || $_GET["age"] ) {
     echo "Welcome ". $_GET['name']. "<br />";
     echo "You are ". $_GET['age']. " years old.";

     exit();
  }
?>
<form action = "<?php <b>$_PHP_SELF</b> ?>" method = "GET">
  Name: <input type = "text" name = "name" />
  Age: <input type = "text" name = "age" />
  <input type = "submit" />
</form>
```

Output

It will produce the following result –

Advantages of GET

Below are some of the advantages of the GET request:

- GET request helps us to save the results of a HTML form.
- GET requests are available to view in the browser history.

- It is easy to use GET method to request the required data.
- The GET request can be used to retrieve information that is identified by the request-URI(Uniform Resource Identifier).

Disadvantages of GET

Below are some of the disadvantages of the GET request:

- GET requests cannot be used to transfer images and files.
- GET requests can only be used to get the data.
- GET requests should not be used to pass credentials such as usernames or passwords.
- The limited length of the URL, limits the use of working with GET requests.
- When using the GET request the browser appends the data to the URL, which is not at all secure.
- Query String value can be easily bookmarked in the GET request.

The POST Method

What is HTTP POST Method?

The POST request is a request that is supported by HTTP. The POST request depicts that a web server also accepts the data in the body of the message. POST is very frequently used by the WWW (World Wide Web) to send user-generated data to the web server.

Features of POST

Below are some of the features of the POST request:

- POST requests take the input from the request body and the query string.
- The data passed using the POST method will not be visible in the query parameters of the browser URL.
- The parameters of the POST requests are not saved in the browser history.
- In POST requests, there is no restriction on the length of data that is to be transmitted.
- POST requests are helpful in sending confidential information like passwords and usernames to the server.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_POST["name"] || $_POST["age"] ) {
    if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
```

```
      die ("invalid name and name should be alpha");
   }
   echo "Welcome ". $_POST['name']. "<br />";
   echo "You are ". $_POST['age']. " years old.";

   exit();
 }
?>
<form action = "<?php <b>$_PHP_SELF</b> ?>" method = "POST">
  Name: <input type = "text" name = "name" />
  Age: <input type = "text" name = "age" />
  <input type = "submit" />
</form>
```

Output

It will produce the following result −

Advantages of POST

Below are some of the advantages of the POST request:

- The POST requests helps to know the resource URI.
- The specification of a new resource location header is ver simple when working with POST requests.
- We can also send a request to accept the entity as a new resource, which is later identified by the URI. This helps to change the resource with the help of POST requests.
- Like the GET request, we can also send user-generated data to the web server.
- POST requests come in very handy when we do not have any idea resource that is to be kept in the URL.
- Use POST requests when you need the server, which controls the URL generation of your resources.

- POST requests are very secure as their requests are not visible in the browser history.
- We can transfer a large amount of data using the POST request method.
- POST requests can keep the data private.
- POST requests are also used to send binary and ASCII data.

Disadvantages of POST

Below are some of the disadvantages of the POST request:

- In POST request, the data sent cannot be saved, because the parameters are not visible in the URL.
- The browser history does not show the POST requests.
- The POST method is not compatible with many firewall setups.
- In POST requests, we cannot use spaces, tabs, carnage returns, etc.
- The POST method consumes a lot of time when working with large binary files.

Parameters	GET	POST
Values in URL	In GET method, values are visible in the URL.	In the POST method, values are not visible in the URL.
Length Limitation	GET has a limitation on the length of the values, generally 255 characters.	POST has no limitation on the length of the values since they are submitted via the body of HTTP.
Performance	GET performs are better compared to POST because of the simple nature of appending the values in the URL.	It has lower performance as compared to the GET method because of time spent in including POST values in the HTTP body.
Supported Data Types	This method supports only string data types.	This method supports different data types, such as string, numeric, binary, etc.
Bookmarking	GET results can be bookmarked.	POST results cannot be bookmarked.
Cacheability	GET request is often cacheable.	The POST request is hardly cacheable.
Browser History	GET Parameters remain in web browser history.	Parameters are not saved in web browser history.

Session Handling: Definition, session_start (), session_id (), session_destroy () session variables.

A session in PHP is a mechanism that allows data to be stored and accessed across multiple pages on a website. When a user visits a website, PHP creates a unique session ID for that user. This session ID is then stored as a cookie in the user's browser (by default) or passed via the URL. The session ID helps the server associate the data stored in the session with the user during their visit.

- PHP sessions are used to maintain state, meaning they allow data to persist as users navigate through a site, which would otherwise be stateless (i.e., each request is independent).
- **Example:** If a user logs in to a website, their login status can be stored in a session variable. As the user moves through different pages, the login status can be checked using the session variable.

How Do PHP Sessions Work?

- **Session Start:** When a user accesses a PHP page, the session gets started with the session_start() function. This function initiates the session and makes the session data available through the $_SESSION superglobal array.
- **Session Variables**: Data that needs to be carried across different pages is stored in the $_SESSION array. **For example**, a user's name or login status can be stored in this array.
- **Session ID:** PHP assigns a unique session ID to every user. This session ID is stored in a cookie in the user's browser by default. The session ID is used to retrieve the user-specific data on each page load.
- **Session Data Storage:** The session data is stored on the server, not the client side. By default, PHP stores session data in a temporary file on the server. The location of this storage is determined by the session.save_path directive in the php.ini file.
- **Session Termination**: Sessions can be terminated by calling session_destroy(), which deletes the session data. Alternatively, a session can be closed using session_write_close() to save the session data and free up server resources.

Note: *PHP sessions allow different PHP pages to share data. By calling session_start() on every page, the same session data becomes accessible across all pages, effectively connecting them during a user's visit.*

How to Use PHP Sessions?

Using PHP sessions involves several key steps: starting a session, storing data in session variables, retrieving data, and eventually destroying the session when no longer needed.

1. Starting a Session

To begin using sessions in PHP, you need to start the session with session_start() at the very beginning of the PHP script. This function ensures that the session is available and creates a unique session ID if it doesn't already exist.

```
<?php
session_start(); // Start the session
?>
```

Note: *Always call session_start() before any HTML output in your PHP script. If you output HTML or whitespace before calling session_start(), it will cause an error.*

2. Storing Data in Sessions

Once the session is started, you can store any information in the $_SESSION superglobal array. This allows you to carry data across different pages on the website.

```
<?php
session_start();
$_SESSION['username'] = 'GFG'; // Store session data
$_SESSION['user_id'] = 123;
?>
```

The username and user ID are stored in the session for use on other pages.

3. Retrieving Session Data

Once data is stored in a session, it can be accessed on any page where the session is started.

```
<?php
session_start();
echo $_SESSION['username']; // Output: GFG
?>
```

You can use the session variables to display user-specific information, check login statuses, and perform various operations.

4. Checking if Session Variables Exist

Before using session data, it's a good practice to check if the session variable exists to avoid errors.

```
<?php
session_start();
if (isset($_SESSION['username'])) {
   echo "Welcome, " . $_SESSION['username'];
} else {
   echo "Please log in.";
```

```
}
?>
```

5. Destroying Sessions

When a session is no longer needed, you can terminate it by using session_destroy(). This function removes all session data from the server. However, it does not automatically unset session variables; you need to manually clear them using unset() if needed.

```
<?php
session_start();
unset($_SESSION['username']); // Remove specific session variable
session_destroy(); // Destroy the session
?>
```

If you want to log out the user, destroying the session will remove all user-specific data and effectively "log them out."

PHP Session Functions

PHP provides several built-in functions to work with sessions. Below are some of the most commonly used functions:

- **session_start():** Starts a session or resumes the current session.

```
session_start(); // Start a session
```

- **$_SESSION:** The $_SESSION superglobal array holds session data. You can store and retrieve session data through this array.

```
$_SESSION['user_id'] = 1; // Store data
echo $_SESSION['user_id']; // Retrieve data
```

- **session_destroy():** Destroys all data registered to a session.

```
session_start();
session_destroy(); // Ends the session
```

- **session_regenerate_id():** Regenerates the session ID to enhance security by avoiding session attacks.

```
session_regenerate_id(true); // Regenerate the session ID
```

Why Use PHP Sessions?

- **Maintaining User State:** In web development, each page request is stateless, meaning the server doesn't remember any previous interaction. Sessions allow you to store and retrieve user data (like login status or shopping cart contents) across multiple pages, making the web experience feel seamless.

- **Secure Data Storage:** Unlike cookies, which store data on the client side (in the browser), sessions store data on the server. This makes sessions more secure for handling sensitive information, as the data is not exposed to the user or tampered with on the client side.
- **Personalized User Experience:** Sessions enable you to personalize user experiences by remembering details such as user preferences, authentication status, and choices made on previous pages. For example, a logged-in user's name can be displayed on every page they visit.
- **E-commerce and Shopping Carts:** For e-commerce websites, sessions are crucial to keep track of items in a shopping cart. Without sessions, the cart would be reset each time the user navigates to a different page, leading to a frustrating experience.
- **Security:** PHP sessions help to prevent unauthorized access. Sensitive data such as authentication tokens or user credentials can be stored securely in session variables, reducing the risk of exposure.

Advantages of PHP Sessions

The advantages of PHP Sessions are mentioned below:

- **Security**: Unlike cookies, which store data on the client side, sessions store data on the server, making them more secure for sensitive information.
- **Data Persistence:** Sessions allow data to persist across multiple pages during a user's visit to a site, making it ideal for tracking user activities like login status, shopping cart contents, etc.
- **Efficiency:** Sessions do not require constant data transfer between the client and server, unlike cookies that send data with each request.
- **Automatic Expiration:** PHP sessions can be configured to automatically expire after a certain time of inactivity, which helps in maintaining session security.

PHP Sessions vs. Cookies

Below is the following difference between PHP Session and PHP Cookies.

Sessions	Cookies
Data is stored on the server.	Data is stored on the client-side (in the browser).
More secure as session data is not stored on the client-side.	Less secure as data is stored on the client-side and can be changed or stolen.

Sessions	Cookies
Sessions usually expire when the browser is closed or after a specified inactivity time.	Cookies can have an expiration date set to stay persistent across browser sessions.