Form

- The <form> tag is used to create an HTML form for user input.
- HTML forms are used to collect user input.
- Form can contain input element like text fields, checkboxes, radio button, submit buttons and more. Forms are used to pass data to a server.
- **Syntax:**
  <form ***parameters***> ***...form elements...*** </form>

Attributes of form

Action

- The action attribute specifies where to send the form-data when a form is submitted.
- **Syntax:** <form action="URL">
- Possible values for URL:
  - An absolute URL - points to another web site (like action = "http://www.example.com/example.htm")
  - A relative URL - points to a file within a web site (like action="example.php")

Enctype

- The enctype attribute specifies how the form-data should be encoded when submitting it to the server.
- **Syntax:** <form enctype="value">
- **Note:** The enctype attribute can be used only if method="post".

Method

- The method attribute specifies how to send form-data (the form-data is sent to the page specified in the action attribute).
- **Syntax:** <form method="get|post">

Value	Description
Get	Default. Appends the form-data to the URL in name/value pairs: URL?name=value&name=value
Post	Sends the form-data as an HTTP post transaction

Name

- The name attribute specifies the name of a form.
- The name attribute is used to reference elements in a JavaScript, or to reference form data after a form is submitted.
- **Syntax:** <form name = "text">
  - **text** - Specifies the name of the form

Target

- The target attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.
- **Syntax:** <form target="_blank|_self|_parent|_top|framename">

Value	Description
_blank	The response is displayed in a new window or tab.
_self	The response is displayed in the same frame (this is default).
_parent	The response is displayed in the parent frame.
_top	The response is displayed in the full body of the window.

HTML Form Controls

Text Box

- **<input type="text">** defines a one-line input field for text input**.**
- Attributes of text box:

Attribute	Description
name	Assigns a name to the given field so that you may reference it later.
maxlength	Specifies the maximum number of character for an input field.
readonly	Specifies that an input field is read only (cannot be changed).
required	Specifies that an input field is required (must be filled out).
size	Specifies the width (in characters) of an input field.
value	Specifies the default value for an input field.

<input type="text" name="username" size="25" maxlength="10">

Password

- **<input type="password">** defines a password field**.**
- Attributes of password:

Attribute	Description
name	Assigns a name to the given field so that you may reference it later.
maxlength	Specifies the maximum number of character for an input field.
required	Specifies that an input field is required (must be filled out).
size	Specifies the width (in characters) of an input field.
value	Specifies the default value for an input field.

<input type="password" name="pass" size="25" maxlength="10">

Text Area

- The <textarea> tag defines a multi-line text input control.
- A text area can hold an unlimited number of characters.

- Attributes of text area:

Attribute	Description
name	Specifies a name for a text area.
cols	Specifies the visible width of a text area.
maxlength	Specifies the maximum number of characters allowed in the text area.
placeholder	Specifies a short hint that describes the expected value of a text area.
readonly	Specifies that a text area should be read-only.
required	Specifies that a text area is required/must be filled out.
rows	Specifies the visible number of lines in a text area.

```
<textarea cols="25" rows="5" maxlength="50" placeholder="Please Enter Comment…">
</textarea>
```

Radio Buttton

- **<input type="radio">** defines a radio button**.**
- Radio buttons let a user select ONE of a limited number of choices.
- Attributes of radio button:

Attribute	Description
name	Assigns a name to the given field so that you may reference it later.
value	Specifies the default value for an input field
checked	Specifies whether a radio button should be checked or not. ○ **true** - The radio button is checked ○ **false** - Default. The radio button is not checked

```
< input type="radio" name="gender" value="male" checked> Male
<input type="radio" name="gender" value="female"> Female
```

Check box

- **<input type="checkbox">** defines a check box**.**
- Check boxes allow for multiple items to be selected for a certain group of choices.
- Attributes of check box:

Attribute	Description
name	Assigns a name to the given field so that you may reference it later.
value	Specifies the default value for an input field.
checked	Specifies whether a check box should be checked or not. ○ **true** - The check box is checked ○ **false** - Default. The check box is not checked

```
<input type="checkbox" name="language" value="java" checked >Java
<input type="checkbox" name="language" value=".net">.Net
<input type="checkbox" name="language" value="c++" checked >C++
```

Dropdown List

- Dropdown list is used to select one option from given list of choice.
- The <select> element is used to create a drop-down list.
- The <option> tags inside the <select> element define the available options in the list.
- Attributes of dropdown list:

Attribute	Description
Name	Defines a name for the drop-down list.
Required	Specifies that the user is required to select a value before submitting the form.
Value	Specifies the value for selected element.
selected	When present, it specifies that an option should be pre-selected when the page loads.

```
<select name="language">
    <option>Gujarati</option>
    <option selected>English</option>
    <option>Hindi</option>
</select>
```

List Box

- List Box is used to select single or multiple option from given list of choice.
- The <select> element is used to create a list box.
- The <option> tags inside the <select> element define the available options in the list.
- Attributes of list box:

Attribute	Description
Name	Defines a name for the list box
Required	Specifies that the user is required to select a value before submitting the form
Value	Specifies the value for selected element
selected	When present, it specifies that an option should be pre-selected when the page loads.
multiple	Specifies that multiple options can be selected at once
size	Defines the number of visible options in a list box.

```
<select name="language" size="3" multiple>
    <option>Gujarati</option>
    <option selected>English</option>
    <option>Hindi</option>
    <option>Telugu</option>
</select>
```

Hidden

- **<input type="hidden">** defines a hidden fields**.**
- Define a hidden field (not visible to a user).
- A hidden field often stores a default value, or can have its value changed by a JavaScript.
- Attributes of hidden fields:

Property	Description
name	Sets or returns the value of the name attribute of the hidden input field
value	Sets or returns the value of the value attribute of the hidden input field

<input type="hidden" name="country" value="India">

Button

- **<input type="button">** defines a button.
- Clickable button, that carried some action when it is clicked.
- Attributes of button:

Property	Description
name	Assigns a name to the given field so that you may reference it later.
value	Determines the text label on the button

<input type="button" name="submit" value="Click me">

Reset Button

- **<input type="reset">** defines a reset button.
- Resets all form values to default values.

<input type="reset">

Submit Button

- **<input type="submit">** defines a button for **submitting** a form to a **form-handler**.
- The form-handler is typically a server page with a script for processing input data.
- The form-handler is specified in the form's **action** attribute.

<input type="submit" value="Save">

Submitting Form values, using $_GET and $_POST method

- There are two ways the browser client can send information to the web server.
  - The GET Method
  - The POST Method
- 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 you 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.**

The GET Method

- 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.
- GET can't be used to send binary data, like images or word documents, to the server.
- It is not secure but fast and quick.
- When you are using GET method in the form collection element the information will be send to the destination file through URL using the concept of **Query String**.
- Collection of names and value pairs is called query string. Name and value separated by ampersand (&) sign.
  name1=value1&name2=value2&name3=value3
- Every query string starts with in URL.
- Data is limited to max length of query string.
  Note: GET should NEVER be used for sending passwords or other sensitive information!

Submitting and accessing form value using GET OR Passing variable through URL

Example

```
<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>
```

welcome_get.php

```
<html>
   <body>
         <?php
            $name=$_GET['name'];
            $email= $_GET['email'];
            echo "welcome ".$name."<br>";
            echo "Your email address is: ".$email;
```

```
            ?>
        </body>
    </html>
```

- The **output** could be something like this:
  Welcome Peter
  Your email address is: Peter94@example.com

The POST Method

- 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).
- POST has **no** limits on the amount of information to send.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.
- The POST method can be used to send ASCII as well as binary data.
- POST may be used for sending sensitive data.
- It is more secured but slower as compared to GET.
- It can post unlimited form variables.

Submitting and accessing form value using POST

Example

```
<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>
```

- When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php".
- The form data is sent with the HTTP POST method.
- To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

welcome.php

```
<html>
    <body>
            <?php
                $name=$_POST['name'];
                $email= $_POST['email'];
                echo "welcome ".$name."<br>";
```

```
            echo "Your email address is: ".$email;
        ?>
    </body>
    </html>
```

- The **output** could be something like this:
  Welcome Peter
  Your email address is: Peter94@example.com

The $_REQUEST variable

- The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.
- The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Example

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

ProcessRequest.php

```
<?php
        echo "Welcome ". $_REQUEST['name']. "<br />";
        echo "You are ". $_REQUEST['age']. " years old.";
?>
```

Output:

Welcome Peter
You are 31 years old.

Difference between GET and POST

	GET	POST
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
History	Parameters remain in browser history	Parameters are not saved in browser history

Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restriction. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

PHP Cookie

- A cookie is a small text file that lets you store a small amount of data on the user's computer.
- They are typically used to keeping track of information such as username that the site can retrieve to personalize the page when user visit the website next time.
- Each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.

Creating Cookie

- In PHP you can create cookie using setcookies() function.
- **Syntax:** setcookie(name, value, expire time, path, domain, secure, httponly)
  - **Name –** specify name of cookie that you want to create.
  - **Value –** value that is associated with the cookie created by you.
  - **Expire time –** Expire time is time when cookie will expire and deleted. It is an optional.
  - **Path –** this specifies the directories for which the cookie is valid.
  - **Domain –** The domain represents the Internet domain from which cookie-based communication is allowed
  - **Secure –** 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: The setcookie() function must appear BEFORE the <html> tag.

Example

CreateCookies.php

```
<?php
    setcookie('cookiename','cookievalue', time()+60*60*24*5);
    echo "Cookie is set";
 ?>
```

Retrieve Cookie/ Accessing Cookies Values

- The PHP $_COOKIE variable is used to retrieve a cookie value.
- In the example below, we retrieve the value of the cookie named "username" and display it on a page.

Example

RetrieveCookies.php

```
<?php
    $cookiename =$_COOKIE[' cookiename'];
    echo "Cookie named:".$cookiename;
?>
```

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

RetrieveCookies.php

```
<?php
    if(isset($_COOKIE[' cookiename ']))
            echo " Cookie ".$_COOKIE[' cookiename ']. " is not set! "  ;
    else
            echo " Cookie ".$_COOKIE[' cookiename ']. " is set! ";
?>
```

Removing Cookie/ Delete Cookie

- When deleting a cookie you should assure that the expiration date is in the past.

DeleteCookies.php

```
<?php
    setcookie(' cookiename ',' cookievalue ', time()-60*60*24*5);
    echo "Cookie is deleted";
?>
```

PHP Session

- 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.

PHP Session Variables

- When you are working with an application, you open it, do some changes and then you close it.
- This is much like a Session. The computer knows who you are. It knows when you start the application and when you end.
- But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.
- A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will

be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

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

Starting Session

- Before you can store user information in your PHP session, you must first start up the session.
- A session is started with the **session_start()** function.
- 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.

Creating session variable

- The correct way to store and retrieve session variables is to use the PHP $_SESSION variable.
- **Syntax:** $_SESSION['variableName']=Value;

Example

SessionDemo.php

```
<html>
<head>
    <title>Session Demo</title>
</head>
<body>
    <form name="formname" method="POST" action="SessionDemoProcess.php">
            User Name:<Input type="text" name="username"/><br/>
            Password:<Input type="password" name="pass"/><br/>
            <Input type="submit" name="submit" value="Save"/>
    </form>
</body>
</html>
```

SessionDemoProcess.php

```
<?php
    Session_start();
```

```
$uname = $_POST['username'];
$pwd = $_POST['pass'];
if($uname=="admin" and $pwd=="123")
{
        $_SESSION['username']=$uname;
        header("location:Welcome.php?un=".$_SESSION['username']);
}
else
{
        echo "enter correct username and password";
}
?>
```

Welcome.php

```
<?php
    Session_start();
    if(isset($_SESSION['username']))
    {
            echo "Welcome:".$_SESSION['username'];
    }
    else
    {
            header("location:SessionDemo.php");
    }
?>
<html>
<head>
    <title>Session Demo</title>
    </head>
    <body>
        <a href="Logout.php">Logout</a>
    </body>
</html>
```

Destroying a Session

- If you 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.
- You can also completely destroy the session by calling the session_destroy() function.
- **Note:** session_destroy() will reset your session and you will lose all your stored session data.

Example

Logout.php

```
<?Php
    Session_start();
    session_destroy();
    header("location:SessionDemo.php");
?>
```

Difference between Cookies and Session

COOKIE	SESSION
Cookies are **client-side** files that contain user information	Sessions are **server-side** files that contain user information
In php **$_COOKIE** super global variable is used to manage cookie.	In php **$_SESSION** super global variable is used to manage session.
You don't need to start Cookie as It is stored in your local machine.	Before using $_SESSION, you have to write session_start(); In that way session will start and you can access $_SESSION variable on that page.
Cookies can have a long lifespan, lasting months or even years.	Sessions have a limited lifespan; they expire when the browser is closed.
Cookies are limited in size depending on each browser's default settings.	Sessions are only limited in size if you limit their size on the server.
Cookies can be disabled if the visitor's browser does not allow them.	Sessions cannot be disabled by the visitor because they are not stored in the browser.
Cookies can be edited by the visitor. (Do not use cookies to store sensitive data.)	Sessions cannot be edited by the visitor.
Session ends when user close his browser.	Cookie ends depends on the life time you set for it