Unit-V Working with Data

HTML FORM element

- The HTML <form> element can contain one or more of the following form elements: (reference from: https://www.w3schools.com/html/html_form_elements.asp)
  - <input>
  - <label>
  - <select>
  - <textarea>
  - <button>

The <input> Element

- One of the most used form element is the <input> element.
- The <input> element can be displayed in several ways, depending on the type attribute.

Example:

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

<h2>The input Element</h2>

<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

The <label> Element

- The <label> element defines a label for several form elements.
- The <label> element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.
- The <label> element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the <label> element, it toggles the radio button/checkbox.
- The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together.

The <select> Element

- The <select> element defines a drop-down list.

- The <option> elements defines an option that can be selected.
- By default, the first item in the drop-down list is selected.
- To define a pre-selected option, add the selected attribute to the option:
- Use the size attribute to specify the number of visible values:
- Use the multiple attribute to allow the user to select more than one value.

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

<h2>Allow Multiple Selections</h2>

<p>Use the multiple attribute to allow the user to select more than one value.</p>

<form action="/action_page.php">
  <label for="cars">Choose a car:</label>
  <select id="cars" name="cars" size="4" multiple>
    <option value="volvo" selected>Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select><br><br>
  <input type="submit">
</form>

<p>Hold down the Ctrl (windows) / Command (Mac) button to select multiple options.</p>

</body>
</html>
```

The <textarea> Element

- The <textarea> element defines a multi-line input field (a text area).

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

<form action="action_page.php">
  <textarea name="message" rows="10" cols="30">The cat was playing in the garden.</textarea>
  <br>
  <input type="submit">
</form>

</body>
</html>
```

The <button> Element

- The <button> element defines a clickable button:

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

<button type="button" onclick="alert('Hello World!')">Click Me!</button>

</body>
</html>
```

HTML Form Input Types

- In HTML <input type=" "> is an important element of HTML form. The "type" attribute of input element can be various types, which defines information field.
- Such as <input type="text" name="name"> gives a text box.
- Following is a list of all types of <input> element of HTML.

type=" "	Description
text	Defines a one-line text input field
password	Defines a one-line password input field
submit	Defines a submit button to submit the form to server
reset	Defines a reset button to reset all values in the form.
radio	Defines a radio button which allows select one option.
checkbox	Defines checkboxes which allow select multiple options form.
button	Defines a simple push button, which can be programmed to perform a task on an event.
file	Defines to select the file from device storage.
image	Defines a graphical submit button.

- HTML5 added new types on <input> element. Following is the list of types of elements of HTML5

type=" "	Description
color	Defines an input field with a specific color.
date	Defines an input field for selection of date.
datetime-local	Defines an input field for entering a date without time zone.
email	Defines an input field for entering an email address.
month	Defines a control with month and year, without time zone.
number	Defines an input field to enter a number.
url	Defines a field for entering URL
week	Defines a field to enter the date with week-year, without time zone.
search	Defines a single line text field for entering a search string.
tel	Defines an input field for entering the telephone number.

PHP Super Global Variables

Variable	Contains
$_GET	Variables provided to a script through the GET method.
$_POST	Variables provided to a script through the POST method.
$_COOKIE	Variables provided to a script through a cookie.
$_FILES	Variables provided to a script through file uploads.
$_SERVER	Contains information such as headers, file paths, and script locations.
$_ENV	Contains any variables provided to a script as part of the server environment.
$_REQUEST	Contains any variables provided to a script via any user input mechanism.
$_SESSION	Contains any variables that are currently registered in a session.

Passing variables between pages through GET, POST and REQUEST

$_GET

- $_GET is a super global variable which is used to collect form data after submitting an HTML form with method="get".
- $_GET can also collect data sent in the URL.

First.html

```
<html>
<head>
<title>My First HTML web page</title>
</head>
<body>
    <FORM name="form1" action="result.php" method="GET">
        Name : <input type="text" name="name"><br>
        Password : <input type="password" name="pass"><br>
        <input type="submit" name="submit1" value="Login"> <br>
        <a href="result.php?name=PHP&pass=get">Click Here</a>
    </FORM>
</body>
</html>
```

- When a user clicks on the link "Click Here" and user click on login, the parameters "name" and "pass" are sent to "result.php", and access their values in "result.php" with $_GET.
- The example below shows the code in "result.php":

result.php

```
<?php
        echo "Your Username = ". $_GET["name"]. "<br/>";
        echo " Password = ".$_GET["pass"];
?>
```

$_POST

- $_POST is a super global variable which is used to collect form data after submitting an HTML form with method="post".
- $_POST is also widely used to pass variables.

First.html

```
<html>
<head>
<title>My First HTML web page</title>
</head>
<body>
    <FORM name="form1" action="result.php" method="GET">
        Name : <input type="text" name="name"><br>
        Password : <input type="password" name="pass"><br>
    </FORM>
</body>
</html>
```

result.php

```
<?php
    echo "Your Username = ". $_POST["name"]. "<br/>";
    echo " Password = ".$_POST["pass"];
?>
```

$_REQUEST

- The $_REQUEST is a super global variable which is work with both get and post methods.

First.html

```
<html>
<head>
<title>My First HTML web page</title>
</head>
<body>
```

```
<FORM name="form1" action="result.php" method="GET">
    Name : <input type="text" name="name"><br>
    Password : <input type="password" name="pass"><br>
</FORM>
</body>
</html>
```

result.php

```
<?php
    echo "Your Username = ". $_REQUEST["name"]. "<br/>";
    echo " Password = ". $_REQUEST["pass"];
?>
```

SESSION

- A session is a way to store information (in variables) to be used across multiple pages.
- When you work 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 or what you do, because the HTTP address doesn't maintain state.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
- So; Session variables hold information about one single user, and are available to all pages in one application.

Start a PHP Session

- A session is started with the session_start () function.
- Session variables are set with the PHP global variable: $_SESSION.

```php
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Get PHP Session Variable Values

- Session variables values get with the PHP global variable: $_SESSION.

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

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Session state is usually stored in a temporary file. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it.

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

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Destroy a PHP Session

- To remove all global session variables and destroy the session, use session_unset() and session_destroy().

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

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();

echo "All session variables are now removed, and the session is destroyed."
?>

</body>
</html>
```

COOKIES

- A cookie in PHP is a small file with a maximum size of 4KB that the web server stores on the client computer.
- They are typically used to keep track of information such as a username that the site can retrieve to personalize the page when the user visits the website next time.
- A cookie can only be read from the domain that it has been issued from.
- Cookies are usually set in an HTTP header but JavaScript can also set a cookie directly on a browser.

Create Cookies with PHP

- A cookie is created with the setcookie () function.

Syntax

- setcookie (name, value, expire, path, domain, secure, httponly);

Note: Only the name parameter is required. All other parameters are optional.

- Name: It is used to set the name of the cookie.
- Value: It is used to set the value of the cookie.
- Expire: It is used to set the expiry timestamp of the cookie after which the cookie can't be accessed.
- Path: It is used to specify the path on the server for which the cookie will be available.
- Domain: It is used to specify the domain for which the cookie is available.
- Security: It is used to indicate that the cookie should be sent only if a secure HTTPS connection exists.

- Httponly: it used to indicate that the cookies should be work in http only or not, the value of this parameter is either true or false. Default value is false.

```
<!DOCTYPE html>
<?php
    setcookie("Auction_Item", "Luxury Car", time() + 2 * 24 * 60 * 60);
?>
<html>
<body>
    <?php
    if (isset($_COOKIE["Auction_Item"]))
    {
        echo "Auction Item is a  " . $_COOKIE["Auction_Item"];
    }
    else
    {
        echo "No items for auction.";
    }
    ?>
    <p>
        <strong>Note:</strong>
        You might have to reload the page
        to see the value of the cookie.
    </p>

</body>
</html>
```

- It is always advisable to check whether a cookie is set or not before accessing its value. Therefore to check whether a cookie is set or not, the PHP isset() function is used. To check whether a cookie "Auction_Item" is set or not

Accessing Cookie Values

- For accessing a cookie value, the PHP $_COOKIE super global variable is used.
- It is an associative array that contains a record of all the cookies values sent by the browser in the current request.
- The records are stored as a list where the cookie name is used as the key.

```
<!DOCTYPE html>
<?php
    setcookie("Auction_Item", "Luxury Car", time() + 2 * 24 * 60 * 60);
?>
<html>
<body>
<?php
    echo "Auction Item is a  " . $_COOKIE["Auction_Item"];
?>
    <p>
        <strong>Note:</strong>
        You might have to reload the page
        to see the value of the cookie.
    </p>

</body>
</html>
```

Deleting Cookies:

- The setcookie() function can be used to delete a cookie.
- If you set the expiration date in past, cookie will be deleted.

```
<!DOCTYPE html>
<?php
    setcookie("Auction_Item", "Luxury Car", time() + 2 * 24 * 60 * 60);
?>
<html>
<body>
    <?php
       setcookie("Auction_Item", "", time() - 60);
    ?>
    <?php
        echo "cookie is deleted"
    ?>
    <p>
        <strong>Note:</strong>
        You might have to reload the page
        to see the value of the cookie.
    </p>

</body>
</html>
```

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

Accessing Relational Databases using PHP

PHP and Database

- ***Mysql*** – popular open-source database management system
- ***PHP*** usually works with ***Mysql*** for web-based database applications
- ***XAMPP*** applications—Web-based applications that use ***Lynux/Windows/Mac***, ***Apache***, *Mysql*, and ***php/pear***

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation

PHP Data Objects (PDO)

What is PDO

- Common interface to any number of database systems.
- Written in C, so you know it's FAST!
- Designed to make use of all the PHP 5.1 features to simplify interface.

Introduction to PDO in PHP

PDO stands for PHP Data Objects, which is a PHP extension providing a consistent method of interacting with databases. It's a database access layer providing a uniform method of access to multiple databases. Here's a short explanation of PDO and its benefits:

PDO (PHP Data Objects): PDO is a database access layer providing a uniform method of access to multiple databases. It provides a data-access abstraction layer, which means that, regardless of which database you're using (MySQL, PostgreSQL, SQLite, etc.), you can interact with the database using the same set of functions.

Benefits of Using PDO:

1. **Database Portability:** PDO allows you to switch between database systems without

changing your code. This means your application can support various databases without major modifications.

2. **Security:** PDO provides prepared statements and parameterized queries, which help prevent SQL injection attacks. Parameters are automatically sanitized, reducing the risk of malicious attacks on your database.
3. **Error Handling:** PDO provides a consistent way to handle errors. It simplifies the process of error detection and handling, making it easier to debug database-related issues.
4. **Performance:** PDO can be faster than some other database access methods in PHP because it can use prepared statements and take advantage of native prepared statement support in certain databases, which can improve performance significantly.
5. **Maintainability:** PDO code tends to be more readable and maintainable than direct SQL queries embedded in the code. It separates the database logic from the application logic, making the codebase cleaner and easier to understand.
6. **Object-Oriented Interface:** PDO provides an object-oriented interface, allowing developers to work with databases using objects and methods, which can lead to more organized and intuitive code.

Databases supported by PDO

1 MySQL

2 PostgreSQL

4 Oracle

5 Firebird

6 MS SQL Server

7 Sybase

8 Informix

9 IBM

10 FreeTDS

11 SQLite

12 Cubrid

13 4D

Example

```
<?php
```

```
$dbHost="localhost";
$dbName="myDB";
$dbUser="root";      //by default root is user name.
$dbPassword="";     //password is blank by default
try{
    $dbConn= new PDO("mysql:host=$dbHost;dbname=$dbName",$dbUser,$dbPassword);
    Echo "Successfully connected with myDB database";
} catch(Exception $e){
echo "Connection failed" . $e->getMessage();
}
?>
```

PDO Functions:

(1) new PDO(): Creates a new PDO instance representing a connection to a database.

Example:

```
$dsn = 'mysql:host=localhost;dbname=mydatabase';
$username = 'username';
$password = 'password';
$pdo = new PDO($dsn, $username, $password);
```

(2) prepare(): Prepares a SQL statement for execution and returns a statement object.

Example:

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username");
```

(3) bindParam() / bindValue(): Binds a parameter to the specified variable name.

Example:

```
$username = 'admin';
$stmt->bindParam(':username', $username);
// OR
// $stmt->bindValue(':username', 'admin');
```

(4) execute(): Executes a prepared statement.

Example:

```
$stmt->execute();
```

(5) fetch(): Fetches a row from a result set.

Example:

```
$row = $stmt->fetch(PDO::FETCH_ASSOC);
echo $row['username'];
```

(6) fetchAll(): Fetches all rows from a result set.

Example:

```
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach ($rows as $row) {
   echo $row['username'];
}
```

(7) rowCount(): Returns the number of rows affected by the last SQL statement.

Example:

```
$rowCount = $stmt->rowCount();
```

(8) lastInsertId(): Returns the ID of the last inserted row or sequence value.

Example:

```
$id = $pdo->lastInsertId();
```

Example

```
try {
   $dsn = 'mysql:host=localhost;dbname=mydatabase';
   $username = 'username';
```

```
    $password = 'password';
    $pdo = new PDO($dsn, $username, $password);

    $stmt = $pdo->prepare("INSERT INTO users (username, email) VALUES (:username, :email)");
    $username = 'admin';
    $email = 'admin@example.com';
    $stmt->bindParam(':username', $username);
    $stmt->bindParam(':email', $email);
    $stmt->execute();

    $id = $pdo->lastInsertId();
    echo "New user inserted with ID: $id";
} catch (PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
```

Error Handling in PDO:

DO provides three error handling modes:

(1) PDO::ERRMODE_SILENT: This mode sets PDO to silently return error codes. You can check for errors by calling **PDO::errorCode()** and **PDO::errorInfo()**.

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
```

(2) PDO::ERRMODE_WARNING: This mode makes PDO throw a PHP warning for errors. It's useful during development but should be avoided in production.

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
```

(3) PDO::ERRMODE_EXCEPTION: This mode throws exceptions when errors occur, allowing you to catch and handle exceptions in your code.

$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

Example of Error Handling with PDO:

```
try {
    // Database connection
    $pdo = new PDO('mysql:host=localhost;dbname=mydatabase', 'username', 'password');

    // Set the PDO error mode to exception
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // SQL query
    $sql = "SELECT * FROM users WHERE id = :id";
    $stmt = $pdo->prepare($sql);

    // Bind parameters and execute the query
    $id = 1;
    $stmt->bindParam(':id', $id, PDO::PARAM_INT);
    $stmt->execute();

    // Fetch results
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    // Handle the fetched data (for example, print it)
    print_r($user);
} catch (PDOException $e) {
    // Handle PDO exceptions
    echo "Error: " . $e->getMessage();
} catch (Exception $e) {
    // Handle other types of exceptions
    echo "Error: " . $e->getMessage();
}
```