Web Programming

James Goudy

November 17, 2025

Contents

0.1 PHP Introduction

0.1.1 Background Information

Original Name: *Personal Home Page*
Now: *PHP: Hypertext Preprocessor*
Created by Rasmus Lerdorf in 1994

Official website: www.php.net
Open Source Project

0.1.2 Why Use PHP

- PHP is to learn and use.
- Written specifically for dynamic web page creation
- It's free
- Multi-platformed
- It's the most popular language in use.

0.1.3 PHP Version

Version 8

- Note Version 4 is not compatible with versions 5+
  - Scripted language
  - There is no version 6
- It often works in conjunction with Javascript (client side)
  - Server Side Program –
- PHP only performs actions on the server.
  - Cross-Platformed
- Runs on Unix, Windows, Apple, and other operating systems

0.1.4 PHP Alternatives

- CGI Scripts (Common Gateway Interface – usually written in Perl)
- ASP.NET – Microsoft and runs on an IIS server
- ColdFusion - Adobe

- JSP – Java Server Pages
- Ruby on Rails

0.1.5 PHP Web Servers

- XAMMP - https://www.apachefriends.org/index.html
- Windows, Linux, OSX
- Note: There are differences between the difference of versions Windows, Linux, and OSX. (mainly on how they treat number lengths coming from MySQL)
- MAMP - http://www.mamp.info/en/
- OSX
- WAMP - http://www.wampserver.com/en/
- Windows

0.1.6 Editors

- NetBeans
- Adobe Dream Weaver
- PSPad, Notepad++
- Textwrangler
- Mac
- VS Code
- Your favorite highlighter

0.1.7 MySQL

MySQL is (as of March 2014) the world's second most[a] widely used open-source relational database management system (RDBMS). It is named after co-founder Michael Widenius's daughter, My. The SQL phrase stands for Structured Query Language.

The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation.

MySQL

0.1.8 MariaDB

MariaDB is a community-developed, commercially supported fork of the MySQL relational database management system (RDBMS), intended to remain free and open-source software under the GNU General Public License. Development is led by some of the original developers of MySQL, who forked it due to concerns over its acquisition by Oracle Corporation in 2009.[6]

MariaDB is intended to maintain high compatibility with MySQL, with library binary parity and exact matching with MySQL APIs and commands, allowing it in many cases to function as drop-in replacement for MySQL. However, new features are diverging.[7] It includes new storage engines like Aria, ColumnStore, and MyRocks.

MariaDB is named after Widenius' younger daughter, Maria.

MariaDB

0.2 PHP Basics

0.2.1 Starting out

PHP files are run on the server. In Apache the are usually placed in a folder within the HTDOCS folder.

PHP code is placed between the

```
<?php

    ....

  ?>
```

This can be placed anywhere - in the head or body and multiple times

0.2.2 Syntax

- Statements end with ;
- PHP statements, key words, classes etc are NOT case sensitive
- Variables ***are*** case sensitive

0.2.3 Comments

```
//  single line
# single line

/* multiple line
Comments
*/
```

0.2.4 Variables

- Start with $
- Followed by a letter or underscore
- Cannot start with a number
- Variables are case sensitive
- No special characters except for underscore _
- Variables are created when a value is assigned to it
- VARIABLES ARE LOOSELY TYPE
  - They can hold anything

```
$salary = 20000;
$_salary = 1000000;
```

0.2.5 Constant

```
//Note constants do not use $ and the convention is to use all caps
//use the define command

        define('PI', 3.1415);
        define('CODEGURU', "Jim Goudy");
```

0.2.6 Variable Scope - Local

```
<?php
 // The variable can only be accessed within the function

function myfunc()
{
 $x=42;    //local scope
 echo (This is value x is $x); //note that the variable is within the string
}

myfunc();
?>
```

0.2.7 Variable Scope - Global

```
<?php
// The variable can only be accessed from outside the function

//global scope
$x=42;

function myfunc()
{
    //note that the variable is within the string
 echo This is value x is $x;
}

myfunc();
?>

Global is also a keyword - supervariable
```

0.2.8 Variable Scope - Static

The variable will retain its value after it's been called by a function.

Usually when a function is called the variables are cleared and reinitialized

```
<?php

function myfunc2()

{

 static $xx=42;  //local scope

 echo This is value x is $x; //note that the variable is within the string

$xx = $xx + 100;

echo("<br>This is value x is $xx<br><hr><br>");

}

myfunc2();
```

```
myfunc2();

?>
```

0.3 Loops and If

0.3.1 Example code

```
<?php

echo "hello php";
print "Hello Amigos";
print("http://localhost/wp/page1");
print("\n");

$output = 0;
$counter = 0;

print("For Loop\n");
for ($counter = 0; $counter <= 20; $counter++) {
    echo("for: ".$counter."\n");
}
$counter = 0;
print("\nWhile Loop\n");
while ($counter < 10) {
    echo("while: ".$counter."\n");
    $counter += 1;
}

$num = 12;

if ($num < 10) {
    echo "This was less than ten";
} else if ($num < 15) {
    echo "This was less than 15";
} else {
    echo "This was greater than 15";
}

?>
```

Purpose: Demonstrate basic output, looping structures, and conditional statements.
Breakdown:

1. **Output Statements:**
   - `echo "hello php";`: Prints the string "hello php" to the output.
   - `print "Hello Amigos";`: Prints the string "Hello Amigos" to the output.
   - `print("http://localhost/wp/page1");`: Prints the URL "localhost/wp/page1" to the output.
   - `print("\n");`: Prints a newline character to the output.
2. **Variable Initialization:**
   - `$output = 0;`: Initializes a variable named `$output` with the value 0.
   - `$counter = 0;`: Initializes a variable named `$counter` with the value 0.
3. **For Loop:**
   - `for ($counter = 0; $counter <= 20; $counter++) { ... }`: Iterates from `$counter` equal to 0 up to 20 (inclusive), incrementing `$counter` by 1 in each iteration. Inside the loop, it prints the current value of `$counter`.

4. **While Loop:**
    - `while ($counter <10) { ... }`: Continues to execute as long as `$counter` is less than 10. Inside the loop, it prints the current value of `$counter` and increments `$counter` by 1.
5. **Conditional Statements:**
    - `if ($num <10) { ... } else if ($num <15) { ... } else { ... }`: Checks the value of `$num` and executes the appropriate code block based on the comparison.

Summary: The code demonstrates the following PHP concepts:

- Output using `echo` and `print`.
- Variable declaration and assignment.
- Looping structures: `for` and `while`.
- Conditional statements: `if`, `else if`, and `else`.

The code effectively illustrates how to use these concepts to perform basic operations and control the flow of execution in a PHP script.

0.4 PHP Arrays

0.4.1 Key ideas

- array()
- count()
- Array Positions
- Printing arrays via for statement
- Associative arrays
- var_dump()
- print_r()
- sorting
  - sort()
  - rsort()
  - asort()
  - arsort()
  - ksort()
  - krsort()

0.4.2 Lecture Code

```
<?php
// Arrays

//Some data
$myVar = 42;

// - - - - - - - - - - - - - - - - Create An Array - - - - - - - - - - - - - - - - - - - - - - - -
echo "<br><br><hr><br><br>";

// Note that we declare array using the array keyword
// Also, since PHP is loosely typed, we can store any type of
// data in the array.  Notice how we have strings, numbers and
// a variable in the same array.

$arrayName = array("Hitch Hikers Guide", 3, $myVar);

//Get the actual number of elements using the count function
echo "Count of array elements<br>".count($arrayName);    //display the count on screen
$arrayCount = count($arrayName); //Store the count in a variable


// - - - - - - - - - - - - - - -Access Array Data By Element Number - - - - - - - - - -
echo "<br><br><hr><br><br>";

//print the elements of the array by specific element
echo $arrayName[0]."<br>";
echo $arrayName[1]."<br>";
echo $arrayName[2]."<br>";


// - - - - - - - - - - Check To See If Variable Is An Array - - - - - - - - - - - - - -
echo "<br><br><hr><br><br>";
echo "Check if variable is array<br><br>";
```

```
///check if the variable is an array
if(is_array($arrayName))
{
 echo(" is an array<br><br>");
}
else
{
 echo(" is not an array<br><br>");
}
echo "<br><br><hr><br><br>";

//Print the array in a for loop
for ($cnt = 0; $cnt < $arrayCount; $cnt++ )
{
 echo ("*".$arrayName[$cnt]."<br>");
}


// - - - - - - - - - - Create an associative array  - - - - - - - - - - - - - - - - - - - - -
echo "<br><br><hr><br><br>";
$asscArray = array("key1" => "the first element",
       "key2" => "the second element",
       "key3" => "the third element");
//add a new key and value to $asscArray
$asscArray["bubba"] = 1000;


//access data via key value
echo ($asscArray["key1"]."<br>");
echo ($asscArray['key3']."<br>");
echo ($asscArray["key2"]."<br>");
echo ($asscArray["bubba"]."<br>");


//print associative array via a foreach loop
foreach($asscArray as $xkey => $xvalue)
{
 echo ($xkey." - - - ".$xvalue."<br>");
}

//print associative array via a foreach loop as a table
echo("<table border = 2>");
foreach($asscArray as $xkey => $xvalue)
{
 echo ("<tr><td>".$xkey."</td><td>".$xvalue."</td></tr>");
}
echo("</table>");

// ********************************************************
echo("Create and associate array Example 2");

$capitals = array(  "myKey" =>  "myvalue",
                    "MT"    =>  "Helena",
```

```
                    "ID"    =>  "Boise",
                    "WY"    =>  "Cheynne",
                    "IL"    =>  "Springfield");
echo("<br><br>");

var_dump($capitals,$aan);
echo("<br><br>");

// Add to the array
$capitals["WA"] = "Seattle";

print_r($capitals);

// put in a table
echo('<table border="2">');
foreach($capitals as $key => $value)
{
    echo("<tr><td>".$key."</td><td>".$value."</td></tr>");
}
echo('</table>');

echo("<br><br><hr><br><br>");


// ********************************************************

// - - - - - - - - - - Create A Two Dimensional Array - - - - - - - - - - - - - - - - - - - - - - -
$product  = array(array("S6E",1250,6),
      array("S6A", 900,6),
      array("S5",  500, 5.7),
      array("S4",   80, 5)
      );

echo ("<br><hr><br>Access Array Data Via Element Number<br><br>");
echo ($product[0][0]." costs $".$product[0][1]." screen = ". $product[0][2]."<br>" );
echo ($product[1][0]." costs $".$product[1][1]." screen = ". $product[1][2]."<br>" );
echo ($product[2][0]." costs $".$product[2][1]." screen = ". $product[2][2]."<br>" );
echo ($product[3][0]." costs $".$product[3][1]." screen = ". $product[3][2]."<br>" );


//Print the two dimensional array using two FOR loops
echo("<br><br>Loop<br><br>");
echo("<table border=1>");
for($row = 0; $row < 4; $row++)
{
 echo("<tr>");
 for($col = 0;$col < 3; $col++)
 {
  echo ("<td>".$product[$row][$col]."</td>" );
 }
 echo("</tr>");

}
```

```
echo("</table>");

// - - - - - - - - - - - -  Create A 2 Dimensional Associative Array  - - - - - - - - - - - - -
echo("<br><br>Multi Associative<br><br>");
$product2  = array(array("Model"=>"S6E","Price"=>1250,"Screen"=>6),
      array("Model"=>"S6A","Price"=> 900,"Screen"=>6),
      array("Model"=>"S5", "Price"=> 500, "Screen"=>5.7),
      array("Model"=>"S4", "Price"=>  80, "Screen"=>5)
      );


//Print 2 Dimensional Array Using For Loop
for($row = 0;$row < 4; $row++)
{
 echo($product2[$row]["Model"]." - - ".$product2[$row]["Price"]." - - ".
  $product2[$row]["Screen"]." <br>");
}

//Print 2 Dimensional Array Using For Loop As A Table
echo("<br><br>");
echo("<table border=1>");
for($row = 0;$row < 4; $row++)
{
 echo("<tr>");
 echo("<td>".$product2[$row]["Model"]."</td><td>".$product2[$row]["Price"]."</td><td>".
  $product2[$row]["Screen"]." </td>");
 echo("</tr>");
}
echo("</table>");

echo ("<br><br><br><br><br>");


// - - - - - - - - - - - - - - - - - - - - - -  Sorting  - - - - - - - - - - - - - - - - - -
//Create Sample Array for sorting
$array3 = array(42, 3, 54, 9, 88, 6, 33, 2, 11, 1);

//print the array with var_dump - shows array, data and data types
var_dump($array3);
echo("<br><br>");

//print the array using print_r - shows the array and the values in the array
print_r($array3);
echo("<br><br>");

//print the array using a for statement
for($i = 0; $i < count($array3); $i++)
{
 echo ($array3[$i]."  ");
}

//Sort the array
sort($array3);
```

```
echo("<br><br>");
//print out the array
for($i = 0; $i < count($array3); $i++)
{
 echo ($array3[$i]."  ");
}

//Reverse the array sort
rsort($array3);

echo("<br><br>");

//Print the reverse display
for($i = 0; $i < count($array3); $i++)
{
 echo ($array3[$i]."  ");
}

echo("<br><br><br><br>");
$product3  = array(array("Model"=>"X6E","Price"=>1250,"Screen"=>6),
      array("Model"=>"S6A","Price"=> 900,"Screen"=>6),
      array("Model"=>"J5B", "Price"=> 500, "Screen"=>5.7),
      array("Model"=>"A4C", "Price"=>  80, "Screen"=>5)
      );


$product3  = array( "X6E"=>1250,
     "S6A"=> 300,
     "J5B"=> 500,
     "A4C"=>  80
      );


asort($product3);

echo("<br><br>Associative Array - value sort<br><br>");
echo("<table border = 2>");
foreach($product3 as $xkey => $xvalue)
{
 echo ("<tr><td>".$xkey."</td><td>".$xvalue."</td></tr>");
}
echo("</table>");

arsort($product3);

echo("<br><br>Associative Array - value Reverse sort<br><br>");
echo("<table border = 2>");
foreach($product3 as $xkey => $xvalue)
{
 echo ("<tr><td>".$xkey."</td><td>".$xvalue."</td></tr>");
}
echo("</table>");
```

```
//zzzzzzzzzzzzzzzzz
ksort($product3);

echo("<br><br>Associative Array - key sort<br><br>");
echo("<table border = 2>");
foreach($product3 as $xkey => $xvalue)
{
 echo ("<tr><td>".$xkey."</td><td>".$xvalue."</td></tr>");
}
echo("</table>");

krsort($product3);

echo("<br><br>Associative Array - key reverse sort<br><br>");
echo("<table border = 2>");
foreach($product3 as $xkey => $xvalue)
{
 echo ("<tr><td>".$xkey."</td><td>".$xvalue."</td></tr>");
}
echo("</table>");
echo ("<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>");
?>
```

0.4.3 Summary of Code

The `if` statement in the provided code is used to check if the variable `$arrayName` is an array. Here's a breakdown of its purpose:

Purpose:

- To verify the data type of the variable `$arrayName`.
- To ensure that subsequent operations on the variable are performed correctly based on its type.

Logic:

- The `is_array($arrayName)` function is used to determine if `$arrayName` is an array.
- If the function returns `true`, it means `$arrayName` is indeed an array. In this case, the code within the `if` block is executed, which prints the message " is an array" to the output.
- If the function returns `false`, it means `$arrayName` is not an array. In this case, the code within the `else` block is executed, which prints the message " is not an array" to the output.

Array Creation and Access:

- **Indexed arrays:** Created using the `array()` keyword and accessed by their numeric index.
- **Associative arrays:** Created using key-value pairs and accessed by their keys.
- **Multidimensional arrays:** Created by nesting arrays within each other and accessed using multiple indexes.

Array Functions:

- `count()`: Returns the number of elements in an array.
- `is_array()`: Checks if a variable is an array.
- `sort()`: Sorts an array in ascending order.
- `rsort()`: Sorts an array in descending order.
- `asort()` : Sorts an associative array by value in ascending order.
- `arsort()`: Sorts an associative array by value in descending order.
- `ksort()` : Sorts an associative array by key in ascending order.
- `krsort()`: Sorts an associative array by key in descending order.

Array Traversal:

- `for` loops: Used to iterate over indexed arrays.
- `foreach` loops: Used to iterate over associative arrays.

Array Output:

- `echo` and `print`: Used to print array elements.
- `var_dump()`: Prints detailed information about an array, including its data type and values.
- `print_r()`: Prints a human-readable representation of an array.

0.5 Functions

0.5.1 Code Breakdown and Comments

Purpose: This PHP code demonstrates various programming concepts, including functions, loops, and conditional statements.

Functions:

- **`myfunc1()`:** A simple function that prints "Func1" to the page
- **`add2($n1, $n2)`:** Takes two numbers as input, adds them together, and returns the sum.
- **`loops()`:** A function that demonstrates different types of loops (while, do-while) and conditional statements.

Variables:

- **`$n1`, `$n2`:** Used in the `add2()` function to store the input numbers.
- **`$sum`:** Stores the calculated sum in the `add2()` function.
- **`$run`, `$cntr`, `$stop`, `$status`:** Variables used in the `loops()` function for loop control, counting, and status tracking.

Loops:

- **`while` loop:** Continues to execute as long as a given condition is true.
- **`do-while` loop:** Executes at least once, then continues as long as a given condition is true.

Conditional Statements:

- **`if-else` statements:** Used to make decisions based on conditions.

Code Breakdown:

1. **Function Definitions:**
   - `myfunc1()` is defined to print a message.
   - `add2()` takes two numbers, adds them, and returns the result.
   - `loops()` demonstrates different loop types and conditional logic.
2. **Main Execution:**
   - Calls the `myfunc1()` function.
   - Calculates the sum using `add2()` and prints the result.
   - Calls the `loops()` function to demonstrate loop and conditional concepts.

Comments within `loops()`:

- **`while` loop:** Demonstrates a basic `while` loop with a counter and a stopping condition.
- **`while` loop with `if`:** Shows how to use an `if` statement inside a loop to break out of the loop based on a condition.
- **`do-while` loop:** Demonstrates a `do-while` loop that executes at least once.
- **`if-else` statements:** Shows how to use `if-else` statements to check conditions and set a status based on the result.

```
<!doctype html>
<html>
    <head>
        <title>functions</title>

        <?php

        function myfunc1()
        {
            echo("<br><br>Func1<br><br>");
        }
```

```
        function add2($n1,$n2)
        {
            $sum = $n1 + $n2;

            return $sum;
        }

        function loops()
        {

            $run = TRUE;

            $cntr = 1;
            $stop = 11;

            // Example of a while loop
            while($cntr < $stop)
            {
                echo($cntr.". Item".$cntr."<br>");
                $cntr +=1;
            }
            echo("<br><hr><br>");
    // excample of while loop using
            // a boolean and if statement
            $cntr = 0;
            while($run)
            {
                echo($cntr.". Item 22 ".$cntr."<br>");
                $cntr +=1;

                if($cntr == $stop)
                {
                    $run=FALSE;
                }
            }

            echo("<br><hr><br>");
            // Example of do loop - always runs once
            $cntr = 0;
            do{
                echo($cntr.".  Item Do".$cntr."<br>");
                $cntr +=1;
            }while($cntr < $stop);


        echo("<br><hr><br><h2>Status If</h2><br>");

        $cntr = 0;
        $status = "";
        $run = True;

        while($run)
```

```
        {

   // example of else if statement
            if($cntr == $stop)
            {
                $run=FALSE;
            }
            else if(($cntr % 2) == 0)
            {
                $status = "Even";
            }
            else{
                $status = "Odd";
            }

            echo($cntr.". Item 22 ".$cntr." ".$status."<br>");
            $cntr +=1;
        }
        }
        ?>

    </head>
<body>
    <?php

        myfunc1();

        echo("<br><br><hr><br><br>");
    ?>
    <h2>Sum Example - 60 + 40</h2>
    <?php
        $x = 40;
        $xx = 60;
        $total = add2($x,$xx);
        echo("The sum = ".$total."<br>");

        echo("<br><br><hr><br><br>");

        loops();
    ?>
</body>
</html>
```

0.6 First Forms - POST and GET

0.6.1 Key Ideas

- POST and GET
- isset()
- empty()
- print_r() and var_dump()
- Use isset() and empty() to write a submit and POST on one PHP page

Definition - GET
GET is used to request data from a specified resource. Note that the query string (name/value pairs) is sent in the URL of a GET request:
demo_form.php?name1=value1&name2=value2
https://www.w3schools.com/tags/ref_httpmethods.asp

Some notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests are only used to request data (not modify)
  https://www.w3schools.com/tags/ref_httpmethods.asp

Definition - POST
POST is used to send data to a server to create/update a resource.
The data sent to the server with POST is stored in the request body of the HTTP request:
https://www.w3schools.com/tags/ref_httpmethods.asp

Some notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

https://www.w3schools.com/tags/ref_httpmethods.asp

0.6.2 Code Examples

Home Page - index.html

```
<! - - index.html - ->
<html>
<head>
<title>Home Page</title>

</head>
<body>
<h1>POST and GET Examples</h1>

<a href="form1.html">Form 1</a><br>
<a href="form2.html">Form 2</a><br>
```

```
<a href="form3.php">Form 3</a><br>
<a href="form4.php">Form 4</a><br>

</body>
</html>
```

First Form - form1.html

```
<! - - form1.html ->
<!DOCTYPE html>
<html>
<head>
<title>Home Page</title>

</head>
<body>

<h1>Form 1</h1>

 <form action="form1a.php" method="POST">
 <table border="1">
  <tr>
  <td>Name</td>
  <td><input type="text" name="stuName" value="Bubba"></td>
  </tr>

  <tr>
  <td>Major</td>
  <td><input type="text" name="stuMajor" value="Whisky Making"></td>
  </tr>


  <tr>
  <td></td>
  <td><input type="submit" value="Click Me"></td>
  </tr>

 </table>
 </form>
</body>
</html>
```

Results from Form1 - form1a.php

```
<! - - form1a.php - ->
<html>
<head>
<title>Form1a</title>
</head>
<body>

<h1> Results from form1</h1>

<?php
```

```
    echo("var dump<br>");
    var_dump($_POST);

    echo("<br><hr><br>");

    print_r($_POST);

    echo("<br><hr><br>");

    echo("<br>Hello ".$_POST['stuName']."<br>");
    echo("Your major is ".$_POST['stuMajor']."<br>");

?>

<br><br>
<a href="index.html">Home</a>
</body>
</html>
```

Form2 (GET) - form1.html

```
<! - - form2.html - ->
<!DOCTYPE html>
<html>
<head>
<title>Home Page</title>

</head>
<body>

<h1>Form 2</h1>

 <form action="form2a.php" method="GET">
 <table border="1">
  <tr>
  <td>Name</td>
  <td><input type="text" name="stuName" value="Brandy">
  </td>
  </tr>

  <tr>
  <td>Major</td>
  <td><input type="text" name="stuMajor" value="Wine Making"></td>
  </tr>


  <tr>
  <td></td>
  <td><input type="submit" value="Click Me"></td>
  </tr>

 </table>
 </form>
</body>
```

```
</html>
```

Results - form2a.php

```
<! - - form2a.php - ->
<html>
<head>
<title>Form2a</title>
</head>
<body>

<h1> Results from form2a</h1>

<?php
    echo("var dump<br>");
    var_dump($_GET);

    echo("<br><hr><br>");

    print_r($_GET);

    echo("<br><hr><br>");

    echo("<br>Hello ".$_GET['stuName']."<br>");
    echo("Your major is ".$_GET['stuMajor']."<br>");
?>

<br><br>
<a href="index.html">Home</a>
</body>
</html>
```

Form and Results - *isset()* and *empty* - form3.php

```
<! - - form3.php - ->
<!DOCTYPE html>
<html>
<head>
<title>Form 3</title>

</head>
<body>

<h1>Form 3</h1>

 <form action="form3.php" method="POST">
 <table border="1">
  <tr>
            <td>Name</td>
            <td><input type="text" name="stuName" value="Joe"></td>
  </tr>

  <tr>
            <td>Major</td>
```

```
            <td><input type="text" name="stuMajor" value="Bagpipe Making"></td>
  </tr>


  <tr>
            <td></td>
            <td><input type="submit" value="Click Me"></td>
  </tr>

 </table>
 </form>

    <?php
        echo("var dump<br>");
        var_dump($_POST);

        echo("<br><hr><br>");

        print_r($_POST);

        echo("<br><hr><br>");

        echo("<h3>isset</h3>");
        if(isset($_POST['stuName']))
        {
            echo("<br>Hello ".$_POST['stuName']."<br>");
            echo("Your major is ".$_POST['stuMajor']."<br>");
        }

        echo("<br><hr><br>");

        echo("<h3>Not Empty</h3>");
        if(!empty($_POST['stuName']))
        {
            echo("<br>Hello ".$_POST['stuName']."<br>");
            echo("Your major is ".$_POST['stuMajor']."<br>");
        }

    ?>

<a href="index.html">Home</a>
</body>
</html>
```

Form4 - form4.php

This is the standard format for doing PHP pages with one page.

```
<! - - form4.php - ->

<!DOCTYPE html>
<html>
<head>
<title>Form 4</title>
```

```
</head>
<body>

<?php

if(!isset($_POST['stuName']) || empty($_POST['stuName'])){
    echo('
    <form action="form4.php" method="POST">
        <table border="1">
            <tr>
                <td>Name</td>
                <td><input type="text" name="stuName" value="Randy"></td>
            </tr>

            <tr>
                <td>Major</td>
                <td><input type="text" name="stuMajor" value="Basket Weaving"></td>
            </tr>


            <tr>
                <td></td>
                <td><input type="submit" value="Click Me"></td>
            </tr>
        </table>
        </form>
    ');
}
else
{
    echo("<br>Hello ".$_POST['stuName']."<br>");
    echo("Your major is ".$_POST['stuMajor']."<br>");
}
?>

<a href="index.html">Home</a>
</body>
</html>
```

0.7 HTML Input Field Review

Input Review

Sample Form

Text Input	Enter full name Full Name
Tel Input	555-555-5555
Text Input / Size and Length	
Title	Mrs. Ms.
Radio	Mr Mrs Ms Dr
Billing Address	Same as billing
Email	
	Submit

Array ([text1] => 33 [tel1] => [text2] => dd [title] => Dr. [rtitle1] => Mrs [ba] => on [email] => a@a.com)

Text box says: 33

```
<!DOCTYPE html>
<html>
    <head>
        <title>Input Review</title>
        <style>
            .center {
                    margin: auto;
                    width: 50%;
                    }
        </style>
    </head>
    <body>
        <div id="div1" class="center">
        <h1>Input Review</h1>
        Sample Form
        <form action="inputreview.php" method="POST">
            <table border="1">
                    <! - -
                    Text input box allows a user to enter text.
                    type="text"
                    placeholder - this show "sample" text
                    name - used to submit a form element to the server
                    id - used to uniquely identify any element in the html page.
                        The id attribute is used with JavaScript and CSS.
                    required - used to check if a field is "blank"
                    pattern - checks to see if data fits a regex pattern
                    title - is the help text for when a pattern fails
                    maxlength - this is the total length the value is
                                allow to have. A length of 2 only allows
                                the value to be 2 characters long
                    size - this is the display size of the input field
                    - ->
                <tr>
                    <td>Text Input</td>
                    <td><input type="text"
```

```
            name="text1" id="text1"
            placeholder="Enter full name"
            >
            <label for="text1">Full Name</label
            </td>
    </tr>

        <! - -
        Tel input box allows a user to enter phone number.
        type="tel"
        placeholder - this show "sample" text
        name - used to submit a form element to the server
        id - used to uniquely identify any element in the html page.
             The id attribute is used with JavaScript and CSS.
        required - used to check if a field is "blank"
        pattern - checks to see if data fits a regex pattern for a phone number
        title - is the help text for when a pattern fails
        - ->
    <tr>
    <td>Tel Input</td>
        <td><input type="tel"
            name="tel1" id="tel1"
            placeholder="555 -555 -5555"
            pattern="[0 -9]{3} -[0 -9]{3} -[0 -9]{4}"
            title= "Please enter a phone number as shown by the demo"
            >
        </td>
    </tr>

        <! - -
        Text input box allows a user to enter text.
        type="text"
        placeholder - this show "sample" text
        name - used to submit a form element to the server
        id - used to uniquely identify any element in the html page.
            The id attribute is used with JavaScript and CSS.
        required - used to check if a field is "blank"
        pattern - checks to see if data fits a regex pattern
        title - is the help text for when a pattern fails
        maxlength - this is the total length the value is
                    allow to have. A length of 2 only allows
                    the value to be 2 characters long
        size - this is the display size of the input field
        - ->

    <tr>
        <td>Text Input / Size and Length</td>
        <td><input type="text"
            name="text2" id="text2"
            maxlength="2"
            size= "5"
            value=""
            required></td>
```

```
</tr>

<! - -
Drop Down Fields
name - used to submit a form element to the server
id - used to uniquely identify any element in the html page.
    The id attribute is used with JavaScript and CSS.
size - is the number of rows to on the dropdown form
the <option value> tag is used to set the drop down values
- ->
<tr>
    <td>Title</td>
    <td>
        <select name="title" id="title" size="2" >
            <option value="Mr." >Mr.</option>
            <option value="Mrs.">Mrs.</option>
            <option value="Ms." selected>Ms.</option>
            <option value="Dr.">Dr.</option>
        </select>
    </td>
</tr>

<! - -
Radio Buttons
Radio buttons can have unique id's.  However, in order to group the buttons
so only one button is selected per group, the same group name is used
in the "name" attribute.  Below is an example of two groups - rtitle1 and
rtitle2
- ->
<tr>
    <td>Radio</td>
    <td>
        <input type="radio" value="Mr" id="rd1" name="rtitle1">
        <label for="rd1">Mr</label><br>

        <input type="radio" value="Mrs" id="rd2" name="rtitle1">
        <label for="rd2">Mrs</label><br>

        <input type="radio" value="Ms" id="rd3" name="rtitle2">
        <label for="rd3">Ms</label><br>

        <input type="radio" value="Dr" id="rd4" name="rtitle2">
        <label for="rd4">Dr</label><br>

        </td>
</tr>

<! - -
Checkbox will evaluate to the value of "on" when checked.
- ->
<tr>
    <td>Billing Address</td>
    <td><input type="checkbox" id="ba" name="ba"><label for="ba">Same as billi
```

```
                </tr>

                <! - -
                Email will check to see if there is an @ sign in the text
                - ->
                <tr>
                    <td>Email</td>
                    <td><input type="email" name="email" id="email"></td>
                <tr>
                    <td></td>
                    <td><input type="submit"</td>
                </tr>

                <tr>
                    <td></td>
                    <td></td>
                </tr>
            </table>
        </form>
</div>

        <br><hr><br>

        <?php
            if(!empty($_POST['text1']))
            {
                print_r($_POST);

                echo("<br><br>Text  box says: " . $_POST['text1']);

            }

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

0.8 Session Variables

0.8.1 Key Ideas

- session_start()
- $_SESSION
- session_destroy()
  - die() / exit()
- unset()

Session Variable is a variable that will store values across different pages

Definition
Session Variable is a variable that will store values across different pages

session_start() is a function that tells the webpage to use and access session variables.
session_destroy() is a function that destroys the session variable and all of its contents
unset() is a function that deletes specific data within a session variable associated with a key
die() or exit() it can output a message and it terminates the running of the script

Note
die() is an alias for exit.

0.8.2 Example Code

Home Page / index.php

```
<! - - index.php - ->
<!DOCTYPE html>
<html>
<head>
<title>Home Page</title>
 <?php
        // To session variable - session start
        // has to be on every page you want to use a session variable
        session_start();
    ?>
</head>
<body>

    <?php
        // assign a values a session variable
        $_SESSION["firstName"] = "Bubba";
        $_SESSION["lastName"]  = "Smith";

        // array review
        $myArray1 = array("Larry","Curly","Moe");
        $myArray2 = array( "CEO" => "Sally",
             "CIO" => "Jim",
             "CFO" => "Suzy",
             "CSO" => "Billy");
```

```
        // different ways to add an item to an array
        array_push($myArray1,"Shemp");
        $myArray1[4] = "Curly Joe";
        $myArray1[count($myArray1)] = "Jimbo";
        $_SESSION["stooges"] = $myArray1;
        $_SESSION["managers"] = $myArray2;
    ?>

<h1>Session Variables</h1>

<h2>Session</h2>
    <?php
        echo('$_SESSION<br>');
        print_r($_SESSION);
        echo("<br><hr><br>");
        print_r($myArray1);
        echo("<br><hr><br>");
        print_r($myArray2);
        echo("<br><hr><br>");
        print_r($myArray1);
        echo("<br><hr><br>");
        var_dump($_SESSION);
        echo("<br><hr><br>");
    ?>

<h2>People</h2>

    <?php
        // pull and individual
        echo($_SESSION["stooges"][1]."<br>");
        echo($_SESSION["managers"]["CFO"]."<br>");

        echo("<br><hr><br>");

        $outStooges =$_SESSION["stooges"];
        print_r($outStooges);

        echo("<h3>Stooges</h3>");
        for($i = 0; $i < count($outStooges); $i++)
        {
            echo($outStooges[$i]."<br>");
        }
        echo("<br><hr><br>");
        for($i = 0; $i < count($_SESSION['stooges']); $i++)
        {
            echo($_SESSION["stooges"][$i]."<br>");
        }
    ?>

<br>
<h2>Pages</h2>
<a href="index.php">Home</a><br>
<a href="page2.php">Page 2</a><br>
```

```
<a href="page3.php">Page 3</a><br>
<a href="page4.php">Destroy</a><br>
<a href="page5.php">Delete managers and first Name</a><br>

<br><br><br><br>
</body>
</html>
```

Page 2 / page2.php

```
<! - - "page2.php - ->
<!DOCTYPE html>
<html>
<head>
<title>Page 2</title>

    <?php
     // start session variables
     // needed on every page that accesses the session variable
  session_start();
 ?>

</head>
<body>
<h1>Page 2</h1>
    <?php
     // purpose here is to show that
     // information stored in the session variable can be
     // accessed on different web pages
        echo($_SESSION["stooges"][1]."<br>");
        echo($_SESSION["managers"]["CFO"]."<br>");

        echo("<br><hr<br>");
     // print the key and the value
        echo("<h2>Managers</h2>");
        foreach($_SESSION["managers"] as $key => $value){
            echo("Title: ".$key."   ".$value."<br>");
        }

    ?>

<br>
<h2>Pages</h2>
<a href="index.php">Home</a><br>
<a href="page2.php">Page 2</a><br>
<a href="page3.php">Page 3</a><br>
<a href="page4.php">Destroy</a><br>
<a href="page5.php">Delete managers and first Name</a><br>

<br><hr><br>
    <?php
     // show the whole session variable
     print_r($_SESSION);
    ?>
```

```
<br><br>

</body>
</html>
```

Page 3 / page3.php

```
<?php
    // start session variables
    // needed on every page that accesses the session variable
    session_start();

 // delete the entire session variable and all of its data
    session_destroy();

 // link to page 2 to show that the data was deleted
    echo('<br><a href="page2.php">Page 2</a>');

?>
```

Page 4 / page4.php

```
<?php
    // start session variables
    // needed on every page that accesses the session variable
    session_start();

 // delete the entire session variable and all of its data
    session_destroy();

 // the header command will automatically take the user
 // back to page 2
 header("Location: page2.php");

 //exit the script
 // NOTE: exit() can also be used instead of die()
    die();

?>
```

Delete Managers and First Name / page5.php

```
<?php
    // start session variables
    // needed on every page that accesses the session variable
    session_start();


    // delete specific data by keys in the session variable
    unset($_SESSION['managers']);
    unset($_SESSION['firstName']);

 // the header command will automatically take the user
 // back to page 2
 header("Location: page2.php");
```

```
 //exit the script
 // NOTE: exit() can also be used instead of die()
    die();

?>
```

0.8.3 Session Variable Homework Assignment Hint

index.php

```
<html>
<head>
<?php
session_start();
?>
</head>
<body>
<?php
 print_r($_SESSION);
 echo('<br><hr><br>');
?>

<form action="page2.php" method="POST">
<table>

  <tr>
  <td>First Name</td>
  <td><input type="text" name="fName"></td>
  </tr>

  <tr>
  <td>Last Name</td>
  <td><input type="text" name="lName"></td>
  <tr>

  <td></td>
  <td><input type="submit" name="submit" value="Submit">
  </td>
  </tr>

  </table>
  </form>

  <br>
  <a href="page2.php">Page2</a>

</body>
</html>
```

page2.php

```
<html>
<head>
<title>page2</title>
<?php
 session_start();
?>
</head>
<body>

<?php
```

```
 print_r($_SESSION);
 echo('<br><hr><br>');
 print_r($_POST);
 echo('<br><hr><br>');

 //1 need to see if there is an array in the session variable
 //  if not, then we create one.

 if(isset($_SESSION['arrPlayer']))
 {
  // we have an session variable array
  // that is keep track of names
  array_push($_SESSION['arrPlayer'],$_POST['fName']);
  array_push($_SESSION['arrPlayer'],$_POST['lName']);
 }
 else
 {
  //There is no array in the session variable
  // so we create one

  // create an array
  $arrPlayer = array($_POST['fName'],$_POST['lName']);

  // add it to the session variable
  $_SESSION['arrPlayer']=$arrPlayer;
 }

 echo('<br><h1>Session contents</h1><br>');
 // NOTE in this example we only have 2 items fname, lname
 $xcntr = 0;

 for($c = 0; $c < count($_SESSION['arrPlayer']); $c++)
 {
  echo($_SESSION['arrPlayer'][$c]." ");

  $xcntr = $xcntr+1;
  // in this example we have two items
  // when the xcntr gets to 2
  // when enter a new line and reset the xcntr
  // back to 0
  if($xcntr == 2)
  {
   echo("<br>");
   $xcntr=0;
  }
 }

?>
<br><hr><br>
<a href="index.php">Home</a>
</body>
</html>
```

0.9 PDO Connections

0.9.1 Introduction

The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. Each database driver that implements the PDO interface can expose database-specific features as regular extension functions. Note that you cannot perform any database functions using the PDO extension by itself; you must use a database-specific PDO driver to access a database server.

PDO provides a data-access abstraction layer, which means that, regardless of which database you're using, you use the same functions to issue queries and fetch data. PDO does not provide a database abstraction; it doesn't rewrite SQL or emulate missing features. You should use a full-blown abstraction layer if you need that facility.

PDO ships with PHP. php introduction 10/2023

0.9.2 Database Connections

Connection strings do the following:

- identify what type of database they are connecting to
- where the database/host is located on the web
- the database name
- username with the appropriate privileges
  - NOTE: when going to production, **No username facing the public internet should have admin rights!**
- password for the username

NONE OF THE EXAMPLES INCLUDE SANITIZING THE DATA. IN PRODUCTION, ALWAYS SANITIZE AND VALIDATE THE DATA BEFORE INSERTING IT INTO A DATABASE!

```
try {

 // MSSQL Server, Sybase, and Oracle

 $pdo = new PDO("sqlsrv:host=$host;dbname=$dbname, $user, $password");
 $pdo = new PDO("sybase:host=$host;dbname=$dbname, $user, $password");
 $pdo = new PDO("oci:host=$host;dbname=$dbname, $user, $password");

 // MySQL with PDO_MYSQL
 $pdo = new PDO("mysql:host=$host;dbname=$dbname", $user, $password);

 // SQLite Database
 $pdo = new PDO("sqlite:my/database/path/database.db");

 // ODBC
 $pdo = new PDO("odbc:host=$host;dbname=$dbname", $user, $password);
}

catch(PDOException $e) {

  echo $e ->getMessage();

}
```

0.9.3 PDO Drivers

- | Driver name | Supported databases | | :———— | :———————————————— | | PDO_CUBRID | Cubrid | | PDO_DBLIB | FreeTDS / Microsoft SQL Server / Sybase | | PDO_FIREBIRD | Firebird | | PDO_IBM | IBM DB2 | | PDO_INFORMIX | IBM Informix Dynamic Server | | PDO_MYSQL | MySQL 3.x/4.x/5.x/8.x | | PDO_OCI | Oracle Call Interface | | PDO_ODBC | ODBC v3 (IBM DB2, unixODBC and win32 ODBC) | | PDO_PGSQL | PostgreSQL | | PDO_SQLITE | SQLite 3 and SQLite 2 | | PDO_SQLSRV | Microsoft SQL Server / SQL Azure | | | From php.org |

0.9.4 Close Connection

```
// close the connection

$pdo = null;
```

0.9.5 Error Modes

PDO::ERRMODE_SILENT

This is the default error mode. It does not show any error exceptions

PDO::ERRMODE_WARNING

This mode will issue a standard PHP warning, and allow the program to continue execution. It's useful for debugging.

PDO::ERRMODE_EXCEPTION

This is the mode you should want in most situations. It fires an exception, allowing you to handle errors gracefully and hide data that might help someone exploit your system. Here's an example of taking advantage of exceptions:

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

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

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

0.9.6 Insert / Update

Prepare ->Sanitize ->Bind -> Execute

0.9.7 Prepared Statements

A prepared statement is a precompiled SQL statement that can be executed multiple times by sending just the data to the server. It has the added advantage of automatically making the **data used in the placeholders safe from SQL injection attacks**.

```
$pdo= $pdo ->prepare("INSERT INTO tbl_names(first_name,last_name) VALUES(:first_name,:last_nam

$pdo ->execute();
```

Placeholders

```
// no placeholders - ripe for SQL Injection!

$STH = $pdo ->prepare("INSERT INTO folks (name, addr, city) values ($name, $addr, $city)");
```

```
// unnamed placeholders

$STH = $pdo ->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?);


// named placeholders

$STH = $pdo ->prepare("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");
```

Unnamed Place Holders

```
// unnamed placeholders

$STH = $pdo ->("INSERT INTO folks (name, addr, city) values (?, ?, ?);

// assign variables to each place holder, indexed 1 -3

$STH ->bindParam(1, $name);

$STH ->bindParam(2, $addr);

$STH ->bindParam(3, $city);


// insert one row

$name = "Moe"

$addr = "1 Wicked Way";

$city = "Arlington Heights";

$STH ->execute();


// insert another row with different values

$name = "Curly"

$addr = "Schmuck Drive";

$city = "Schaumburg";

$STH ->execute();
```

Use an array

```
// the data we want to insert

$data = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');
```

```
$STH = $pdo ->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?);

$STH ->execute($data);
```

Named Place Holders

```
// the first argument is the named placeholder name - notice named

// placeholders always start with a colon.

$pdo ->bindParam(':name', $name);


// the data we want to insert
$data = array( 'name' => 'Cathy', 'addr' => '9 Dark and Twisty', 'city' => 'Cardiff' );


// the shortcut!

$pdo = $pdo ->("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");

$STH ->execute($data);
```

0.9.8 Fetching Data

PDO::FETCH_ASSOC: returns an array indexed by column name
PDO::FETCH_BOTH (default): returns an array indexed by both column name and number
PDO::FETCH_BOUND: Assigns the values of your columns to the variables set with the ->bindColumn() method
PDO::FETCH_CLASS: Assigns the values of your columns to properties of the named class. It will create the properties if matching properties do not exist
PDO::FETCH_INTO: Updates an existing instance of the named class
PDO::FETCH_LAZY: Combines PDO::FETCH_BOTH/PDO::FETCH_OBJ, creating the object variable names as they are used
PDO::FETCH_NUM: returns an array indexed by column number
PDO::FETCH_OBJ: returns an anonymous object with property names that correspond to the column names

FETCH_ASSOC

This fetch type creates an associative array, indexed by column name

```
// using the shortcut ->query() method here since there are no variable

// values in the select statement.
$STH = $pdo ->query('SELECT name, addr, city from folks');

// setting the fetch mode
$STH ->setFetchMode(PDO::FETCH_ASSOC);

while($row = $STH ->fetch()) {

  echo $row['name'] . "\n";

  echo $row['addr'] . "\n";
```

```
  echo $row['city'] . "\n";

}
```

FETCH_OBJ

This fetch type creates an object of std class for each row of fetched data. Here's an example:

```
// creating the statement
$STH = $DBH ->query('SELECT name, addr, city from folks');


// setting the fetch mode
$STH ->setFetchMode(PDO::FETCH_OBJ);

// showing the results
while($row = $STH ->fetch()) {

  echo $row ->name . "\n";

  echo $row ->addr . "\n";

  echo $row ->city . "\n";

}
```

0.9.9 Last Insert ID

The ->lastInsertId() method is always called on the database handle, not the statement handle, and will return the auto-incremented id of the last inserted row by that connection.

0.9.10 Delete Files

```
$pdo ->exec('DELETE FROM folks WHERE 1');
```

0.10 PHP Closing Tags: Leave Them Out Or Out

The Short Answer

If your file contains *only PHP code*, it's best practice to **omit the closing `?>` tag**.

0.10.1 Why Leave It Out?

The reason is surprisingly simple:

- Any whitespace, newlines, or hidden characters (like a BOM) after a closing `?>` tag are sent directly to the browser as output.

> **Note**
>
> **BOM** stands for **Byte Order Mark** - It's a special, invisible sequence of bytes (`EF BB BF`) that can appear at the **very beginning of a text file** when it's saved with **UTF-8 encoding with BOM**.
> Its original purpose was to tell programs: *"Hey, this file is encoded in UTF-8."*

- This can trigger the infamous **"headers already sent"** error, break JSON APIs, or cause strange blank lines in your HTML.

Frameworks like **Laravel**, **WordPress**, and even the **official PHP manual** recommend leaving it out for clean, bug-free code.

0.10.2 When to Keep It

There are cases where you still need the closing tag:

- If your file mixes **PHP and HTML** (e.g., a template file), the interpreter needs to know when to leave PHP mode and return to raw HTML.
- Example:

```
<?php echo "Hello, world!"; ?>
<p>This is plain HTML again.</p>
```

But in files that contain only PHP logic—like **database connection scripts**, **configuration files**, or **class libraries**—closing tags are unnecessary and risky.

0.10.3 Wisdom Bite: The Door Analogy

Think of `?>` as leaving a **door open** at the end of your script.

- With the door open, stray characters, invisible spaces, or editor quirks can slip in.
- Leaving the tag out keeps the door shut and your script safe from accidental leaks.

0.10.4 The Role of ?> in PHP

- The closing tag is **optional** in PHP-only files.
- The interpreter automatically stops parsing at the end of the file.
- The **official recommendation**: leave the closing tag off in pure PHP files.

0.10.5 Why It Matters with `require` and `include`

When you include a file, whatever is in that file is dropped straight into your running script.

- **With ?> at the end**: If there's an extra space, newline, or BOM after it, PHP treats it as literal output.
  - Headers may fail ("headers already sent").
  - JSON or XML output can become invalid.
  - HTML may display stray whitespace.
- **Without ?>**: The file ends cleanly, nothing slips through, and you avoid hidden output entirely.

That's why **best practice is to omit ?> in PHP-only files**, especially those that are required or included.

0.10.6 Before & After Example (Great for Class)

Problematic file (`dbconnect.php`):

```
<?php
$pdo = new PDO("mysql:host=localhost;dbname=test", "user", "pw");
?>
```

(But the editor sneaks in a newline here)

Main script:

```
<?php
require 'dbconnect.php';
header("Content -Type: application/json");
echo json_encode(["status" => "ok"]);
```

Result: "Headers already sent" error.

Fixed file (`dbconnect.php`):

```
<?php
$pdo = new PDO("mysql:host=localhost;dbname=test", "user", "pw");
```

No closing tag → *notrailingwhitespace* → *cleanoutput*.

0.10.7 Best Practice (Takeaway)

- **Always omit ?> in PHP-only files.**
- Use it only when you must return to HTML mode.
- This avoids accidental output and saves you from hard-to-track bugs.

0.10.8 PDO - First Data Insert - Car Example

Key Ideas

Five Steps Of Inserting Data

1. Create the the insert SQL statement with placeholders (":").
2. Prepare the SQL statement.
3. Save the $_POST data to variables while sanitizing it.
4. Bind the placeholders with the variables of step 3.
5. Execute the SQL statement.

Connection String

> A database connection string is a string that contains the information needed to connect an application to a database. The string includes parameters such as - Server instance, Database name, Authentication details, Username, Password .

dbconnect.php

```
<! - -  dbconnect.php - - ->
<?php
/*
 - - Database - SQL Script
DROP DATABASE IF EXISTS wp_cars;

 - - CREATE DATABASE
CREATE DATABASE IF NOT EXISTS wp_cars;

 - - Use Database
USE wp_cars;

CREATE TABLE `wp_cars`.`tbl_car`(
    `OwnerFN` VARCHAR(20),
    `OwnerLN` VARCHAR(20),
    `Make` VARCHAR(20),
    `Model` VARCHAR(20),
    `Color` VARCHAR(20),
    `CarYear` CHAR(4),
    `Car_Id` INT(6) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY(`Car_Id`)
) ENGINE = InnoDB;


*/

//connect database
try
{
 $pdo = new PDO('mysql:host=127.0.0.1;dbname=wp_cars','root','');
 $pdo ->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
```

```
 //only educational purposes
 $dbstatus = "Good database connection";

}
catch(PDOException $e)
{
 $dbstatus = "Database connection failed<br>".
    $e ->getMessage();

 die();
}
SESSION_START();

?>
```

default.php

```
<?php
// page3.php

// require - all or nothing - will not tolerate an error
// include - will still try to run if there is an error

require('dbconnect.php');

// educational purposes / development purposes
echo($dbstatus."<br><hr><br>");

?>
<!DOCTYPE html>
<html>
<head>
 <title>Page 1</title>
 <style>
        .divpage1 {
          width: 50%;
          background -color:cornsilk;
          margin -left: auto;
          margin -right: auto;
        }
 </style>
</head>
<body>
<div class="divpage1">

<?php

function sanitize($value){

    // strip of any excess white spaces on the ends
    $value = trim($value);

    // get rid of any html or php tags
    $value = strip_tags($value);
```

```
    // convert special characters
    $value = htmlspecialchars($value,ENT_QUOTES,'UTF -8');

    // return the value
    return $value;

}

// alternative
function sanitize2($value){
    return htmlspecialchars(strip_tags(trim($value)),ENT_QUOTES,'UTF -8');
}


if(!isset($_POST['OwnerFN']))
{
    echo('
  <h1>Car Entry - form uses named place holders</h1>
        <form method="POST" action="default.php">
        <table border="1">
            <tr>
                <td>First Name</td>
                <td><input type="text" width="20" name="OwnerFN"
        required value="John"></td>
            </tr>

            <tr>
                <td>Last Name</td>
                <td><input type="text" width="20" name="OwnerLN"
        value="Doe"></td>
            </tr>

            <tr>
                <td>Make</td>
                <td><input type="text" width="20" name="Make"
        value="Ford"></td>
            </tr>

            <tr>
                <td>Model</td>
                <td><input type="text" width="20" name="Model"
        value="F150"></td>
            </tr>

            <tr>
                <td>Year</td>
                <td><input type="text" width="20" name="Year"
        value="1970"></td>
            </tr>

            <tr>
                <td>Color</td>
```

```
            <td><input type="text" width="20" name="Color"
    value="Red"></td>
        </tr>

        <tr>
            <td></td>
            <td><input type="submit" value="Enter"></td>
        </tr>

    </table>
    </form>
');

}
else
{

    try
    {



    //Step 1 - create sql statement

    $sql_insert = "INSERT INTO tbl_car"
            ."(OwnerFN,OwnerLN,Make,Model,CarYear,Color) "
            ."VALUES(:OwnerFN,:OwnerLN,:Make,:Model,:Year,:Color)";

    //Step 2 - prepare our sql statement
    //This will box our information at the ":" placeholders

    $sqlp_insert = $pdo ->prepare($sql_insert);


    //Step 3 - Sanitize the information - this prevents SQL Injection

    $OwnerFN = sanitize($_POST['OwnerFN'] ?? '');
    $OwnerLN = sanitize($_POST['OwnerLN'] ?? ''G);
    $Make = sanitize($_POST['Make'] ?? '');
    $Model = sanitize($_POST['Model'] ?? '');
    $Year = sanitize($_POST['Year'] ?? '');
    $Color = sanitize($_POST['Color'] ?? '');

    //Step 4 - bind our boxes in the sql statement with our variables
    //bind parameters
    $sqlp_insert ->bindparam(":OwnerFN",$OwnerFN);
    $sqlp_insert ->bindparam(":OwnerLN",$OwnerLN);
    $sqlp_insert ->bindparam(":Make",$Make);
    $sqlp_insert ->bindparam(":Model",$Model);
    $sqlp_insert ->bindparam(":Year",$Year);
    $sqlp_insert ->bindparam(":Color",$Color);
```

```
        //Step 5 - Execute the query
        $sqlp_insert ->execute();

        echo("<br>### input was successful ###<br><br><hr><br>");

        // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


        //Display our last car entered

        $sql_selectLastCar =  "SELECT * " .
                        "FROM tbl_car " .
                        "WHERE Car_Id = (SELECT MAX(Car_Id) ".
        "FROM tbl_car )";

        //Run the Query
        $dataSet = $pdo ->query($sql_selectLastCar);

        //loop through the columns in the row
        foreach($dataSet as $row)
        {
        echo($row['OwnerFN']." ".$row['OwnerLN']."<br>");
        echo($row['Make']." ".$row['Model']." ".$row['CarYear']."<br>");
        echo($row['Color']." ".$row['Car_Id']."<br><hr><br>");
        }

        //need to clear the $_POST because of the isset in if statement
        unset($_POST['OwnerFN']);
    }
    catch(PDOException $e)
    {
        echo("**** Input Error ****<br>".$e);
    }

    //add a button so we can add another car
    echo('<a href="default.php"><button type="button">'.
    'Add Another Car</button>');

}
?>
<br><hr><br>
    <br><a href="default.php"><button type="button">Home</button></a>  
    <a href="page2.php"><button type="button">Page 2</button></a>  
    <a href="page3.php"><button type="button">Page 3</button></a>  
<br><br><br>
</div>
</body>
</html>
```

Page2.php

```
<?php
 // page2.php
 require('dbconnect.php');
```

```
?>
<!DOCTYPE html>
<html>
<head>
 <title>Inventory</title>
 <style>
        #div1 {
          width: 75%;
          background -color:cornsilk;
          margin -left: auto;
          margin -right: auto;
        }
 </style>
</head>
<body>
<div id="div1">
 <H1>Car Inventory</H1>

 <table align="center" border="1">
 <tr>
  <td>First Name</td>
  <td>Last Name</td>
  <td>Make</td>
  <td>Model</td>
  <td>Year</td>
  <td>Color</td>
  <td>Car_Id</td>
 </tr>
 <?php
  $sql = "SELECT * FROM tbl_car";
  $ds = $pdo ->query($sql);

  foreach($ds as $row)
  {
   echo('<tr>');
   echo('
    <td>'.$row['OwnerFN'].'</td>
    <td>'.$row['OwnerLN'].'</td>
    <td>'.$row['Make'].'</td>
    <td>'.$row['Model'].'</td>
    <td>'.$row['CarYear'].'</td>
    <td>'.$row['Color'].'</td>
    <td>'.$row['Car_Id'].'</td>
    ');
   echo('</tr>');
}
 ?>
 </table>


<br><hr><br>
    <br><a href="default.php"><button type="button">Home</button></a>  
    <a href="page2.php"><button type="button">Page 2</button></a>  
```

```
    <a href="page3.php"><button type="button">Page 3</button></a>  
<br><br><br>
</div>
</body>
</html>

<?php
// page3.php

// require - all or nothing - will not tolerate an error
// include - will still try to run if there is an error

require('dbconnect.php');

// educational purposes / development purposes
echo($dbstatus."<br><hr><br>");

?>
<!DOCTYPE html>
<html>
<head>
 <title>Page 3</title>
 <style>
        .divpage3 {
          width: 75%;
          background -color:cornsilk;
          margin -left: auto;
          margin -right: auto;
        }
 </style>
</head>
<body>
<div class="divpage3">
<?php

if(!isset($_POST['OwnerFN']))
{
    echo('
    <h1>Car Entry - form uses ? for place holders</h1>
    <form method="POST" action="page3.php">
    <table border="1" align="center">
        <tr>
            <td>First Name</td>
            <td><input type="text" width="20" name="OwnerFN" required value="Roger"></td>
        </tr>
        <tr>
            <td>Last Name</td>
            <td><input type="text" width="20" name="OwnerLN" value="Smith"></td>
        </tr>
        <tr>
            <td>Make</td>
            <td><input type="text" width="20" name="Make" required value="AMC"></td>
        </tr>
        <tr>
```

```
            <td>Model</td>
            <td><input type="text" width="20" name="Model" required value="Gremlin"></td>
        </tr>
        <tr>
            <td>Year</td>
            <td><input type="text" width="20" name="CarYear" required value="1970"></td>
        </tr>
        <tr>
            <td>Color</td>
            <td><input type="text" width="20" name="Color" required value="Green"></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="Enter"></td>
        </tr>
    </table>
    </form>
    ');

}
else
{
    echo('');
 print_r($_POST);
 echo("<br><hr><br>");
 try
 {
 //Step 1 - create sql statement
 $sql_Insert =   "INSERT INTO tbl_car".
    "(OwnerFN,OwnerLN,Make,Model,CarYear,Color) ".
    "VALUES(?,?,?,?,?,?)";

 //Step2 - prepare our sql statement
 //This will box our information at the ":" placeholders

 $sql_Insert =  $pdo ->prepare($sql_Insert);


 //Step 3 - Sanitize the information
 // use filter_var
 $OwnerFN = filter_var($_POST['OwnerFN'],FILTER_SANITIZE_STRING);
 $OwnerLN = filter_var($_POST['OwnerLN'],FILTER_SANITIZE_STRING);
 $Make = filter_var($_POST['Make'],FILTER_SANITIZE_STRING);
 $Model = filter_var($_POST['Model'],FILTER_SANITIZE_STRING);
 $CarYear = filter_var($_POST['CarYear'],FILTER_SANITIZE_STRING);
 $Color = filter_var($_POST['Color'],FILTER_SANITIZE_STRING);



 //Step 4 - Bind our placeholders to our clean variables

 //
 // $sql_Insert ->bindparam(1,$OwnerFN);
```

```
// $sql_Insert ->bindparam(2,$OwnerLN);
// $sql_Insert ->bindparam(3,$Make);
// $sql_Insert ->bindparam(4,$Model);
// $sql_Insert ->bindparam(5,$CarYear);
// $sql_Insert ->bindparam(6,$Color);

$data = array($_POST['OwnerFN'],
   $_POST['OwnerLN'],
   $_POST['Make'],
   $_POST['Model'],
   $_POST['CarYear'],
   $_POST['Color'],);

$data = array($OwnerFN,$OwnerLN,$Make,$Model,$CarYear,$Color);


//Step 5 - Execute the SQL Statement
$sql_Insert ->execute($data);
//$sql_Insert ->execute();
echo("<br>### input was successful # # # <br><hr><br>");

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

$sql_SelectLastCar =
          "SELECT * ".
    "FROM tbl_car ".
    "WHERE Car_Id = ".
    "(SELECT MAX(Car_Id) FROM tbl_car) ";

// run the query
$dataSet = $pdo ->query($sql_SelectLastCar);

// loop the columns
foreach($dataSet as $row)
{
echo($row['OwnerFN']." ".$row['OwnerLN']."<br>");
echo($row['Make']." ".$row['Model']." ".$row['CarYear']."<br>");
echo($row['Color']." ".$row['Car_Id']."<br>");
}

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

unset($_POST);

}
catch(PDOExcept $eio)
{
 echo("**** Input Error<br>".$eio);
}

// add a button
echo('<a href="default.php"><button type="button">'.
'Add Another Car</button></a>');
```

```
}
?>
<br><hr><br>
    <br><a href="default.php"><button type="button">Home</button></a>  
    <a href="page2.php"><button type="button">Page 2</button></a>  
    <a href="page3.php"><button type="button">Page 3</button></a>  
<br><br><br>
</div>
</body>
</html>
```

10/2025

0.10.9 PDO With No Named Placeholders

This example shows how to use SQL with no named placeholders.

```
<!DOCTYPE html>
<html>
<head>
<title>function example</title>

 <! - - css styling - ->
 <style>
  #div1{
   background -color:cornsilk;
   width:65%;
   margin -left:auto;
   margin -right:auto;
  }
 </style>

</head>
<body>
    <?php

        //database connection
        function newDB(){
            //returns a database object
            return new PDO('mysql:host=127.0.0.1;dbname=wp_cars','root','');
        }

        function pdoInsert(){

            // open new datbase connection
            $pdo = newDB();

            // instantiate variables
            // remember the don't come into existance
            // until a value has been assigned to them

            $ownerFN  = "Bubba";
            $ownerLN = "Smith";
            $make  = "Scion";
            $model  = "XB";
            $carYear = rand(2000,2012);
            $color  = "Blue";

            try
            {
                // SQL Statement
                $sql_stmt = "INSERT INTO tbl_car(OwnerFN,OwnerLN,make,model,carYear,color)
                        VALUES (?,?,?,?,?,?)";

                // prepare
                $pdo = $pdo ->prepare($sql_stmt);
```

```
            // bind our parameters
            $pdo ->bindParam(1,$ownerFN);
            $pdo ->bindParam(2,$ownerLN);
            $pdo ->bindParam(3,$make);
            $pdo ->bindParam(4,$model);
            $pdo ->bindParam(5,$carYear);
            $pdo ->bindParam(6,$color);

            // generate some database
            // normally where you assign $_POST variable
            // which has been sanitized to variable

            // for this example auto generate some database

            $numRecs = rand(5,10);

            for($i = 0; $i < $numRecs; $i++)
            {
                $arn   = rand(0,1000);

                // normally the $_POST variable is
                // sanitized and assigned a local variable

                // $ownerFN = filter_var($_POST['ownerFN'],FILTER_SANITIZE_STRING);

                $ownerFN  = "Bubba".$arn;
                $ownerLN  = "Smith".$arn;
                $make   = "Ford".$arn;
                $model   = "Mach E".$arn;
                $carYear  = rand(2020,2024);
                $color   = "red";

                // since the variables are already bound
                // we can just call execute()

                $pdo ->execute();
            }

        }
        catch(PDOException $epdo)
        {
            // In production, errors be redirect to an error page
            echo($pdo ->getMessage());
        }

        // always closse our connection
        $pdo = null;
    }

    function printCarTable(){

        echo('
        <div id="div1">
```

```
        <h1>Car Table</h1>
        <a href="default.php">
            <button type="Button">Refresh</button>
            </a><br><br>
        <table border="1" width="100%" align="center">
            <tr>
                <td>Owner FN</td>
                <td>Owner LN</td>
                <td>Make</td>
                <td>Model</td>
                <td>Car Year</td>
                <td>Color</td>
                <td>Car ID</td>
            </tr>
        ');

        // create a database connection
        $pdo = newDB();

        // setup our SQL
        $sql = "SELECT * FROM tbl_car";

        // executing the SQL - bring down the data
        $dataSet = $pdo ->query($sql);

        // iterate through each dataSet row
        foreach($dataSet as $row)
        {
            echo('<tr>');
            echo('<td>'.$row['OwnerFN'].'</td>'.
                    '<td>'.$row['OwnerLN'].'</td>'.
                    '<td>'.$row['Make'].'</td>'.
                    '<td>'.$row['Model'].'</td>'.
                    '<td>'.$row['CarYear'].'</td>'.
                    '<td>'.$row['Color'].'</td>'.
                    '<td>'.$row['Car_Id'].'</td>'
                );
            echo('</tr>');

        }

        echo('</table>
        <br></div><br><br>');

        // close the db connection
        $pdo = null;

    }


    function clearDB()
    {
        try
```

```
            {
                // create new database connection
                $pdo = newDB();

                // execute the truncate SQL command
                $pdo ->exec("TRUNCATE TABLE tbl_car");

            }
            catch(PDOException $epdo)
            {
                // In production, errors be redirect to an error page
                echo("Error<br>".$epdo ->getMessage());
            }

            $pdo = null;
        }

        // function calls

        clearDB();
        pdoInsert();
        printCarTable();

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

0.10.10 String Sanitation - filter_var()

This code demonstrates the different sanitation conditions.

In order to see the htmlspecialcharacters, you must look at the source code. There one will see that the special characters have been encoded.

Lecture Code

```
<!DOCTYPE html>
<html>
<head>
<title>PHP Sanitization</title>

    <style>
        main,header,footer{
            margin -left: auto;
            margin -right: auto;
            width: 80%;
        }
    </style>
</head>
<body>

  <header>
    <h1>Sanitization Examples</h1>
  </header>

  <main>

  <! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

  <?php


$testString = "ASDasd1234.99!@#$%^&*()_+ -,./\";\’<>?:[]{}\|‘~ ";
$testString2 = "<h1><p>This is a test</p></h1>";
$testString_Int = "+ - 123,456,789";
$testString_Float = "+ - 123,456,789.8899";

echo(’<font face="Courier New">’);

echo(’<h2>’);
echo("Test Strings<br>");
echo($testString."<br>");
echo($testString_Int."<br>");
echo($testString_Float."<br>");
$test1 = $testString;
$space = "     ";

echo(’<br><hr><br>’);
echo(" FILTER_SANITIZE_EMAIL<br>");
$test1 = filter_var($test1,FILTER_SANITIZE_EMAIL);
echo($testString.’<br>’);
echo($test1.$space."  ");
```

```
echo('<br><hr><br>');


$test1 = $testString_Float;
echo('FILTER_SANITIZE_NUMBER_FLOAT<br>');
$test1 = filter_var($test1,FILTER_SANITIZE_NUMBER_FLOAT);
echo($testString_Float.'<br>');
echo($test1.$space." ");
echo('<br><hr><br>');

$test1 = $testString_Float;
echo('FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND<br>');
$test1 = filter_var($test1,FILTER_SANITIZE_NUMBER_FLOAT,FILTER_FLAG_ALLOW_THOUSAND);
echo($testString_Float.'<br>');
echo($test1.$space." ");
echo('<br><hr><br>');

$test1 = $testString_Float;
echo('FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION<br>');
$test1 = filter_var($test1,FILTER_SANITIZE_NUMBER_FLOAT,FILTER_FLAG_ALLOW_FRACTION);
echo($testString_Float.'<br>');
echo($test1.$space." ");
echo('<br><hr><br>');

$test1 = $testString_Float;
echo('FILTER_SANITIZE_NUMBER_FLOAT,
        FILTER_FLAG_ALLOW_THOUSAND | FILTER_FLAG_ALLOW_FRACTION<br>');
$test1 = filter_var($test1,FILTER_SANITIZE_NUMBER_FLOAT,
                    FILTER_FLAG_ALLOW_THOUSAND | FILTER_FLAG_ALLOW_FRACTION);
echo($testString_Float.'<br>');
echo($test1.$space." ");
echo('<br><hr><br>');

$test1 = $testString_Int;
echo('FILTER_SANITIZE_NUMBER_INT<br>');
$test1 = filter_var($test1,FILTER_SANITIZE_NUMBER_INT);
echo($testString_Int.'<br>');
echo($test1.$space."  ");
echo('<br><hr><br>');



$test1 = $testString;
echo('FILTER_SANITIZE_SPECIAL_CHARS<br>');
$test1 = filter_var($test1,FILTER_SANITIZE_SPECIAL_CHARS,
                array('flags'=>  FILTER_FLAG_STRIP_HIGH  ));
echo($testString.'<br>');
echo($test1.$space."  ");
echo('<br><hr><br>');

$test1 = $testString;
echo('FILTER_SANITIZE_STRING<br>');
$test1 = filter_var($test1,FILTER_SANITIZE_STRING);
```

```
echo($testString.'<br>');
echo($test1.$space."  ");
echo('<br><hr><br>');
//echo('<br><br><hr><br>');

$test1 = $testString;
echo('<br>');
$test1 = htmlspecialchars($test1,ENT_COMPAT);
//$test1 = filter_var($test1,FILTER_SANITIZE_STRING);
echo($test1.$space." htmlspecialchars");


echo('</h2>');
echo('</font');

?>

 <! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  </main>

  <footer>
    <p>&copy; 2023 Jim Goudy</p>
  </footer>

</body>
</html>
```

Output

Sanitization Examples Test Strings ASDasd1234.99!@#$%&̂*()_+-,./";'<>?:[]{}|'~ +- 123,456,789 +- 123,456,789.8899

FILTER_SANITIZE_EMAIL ASDasd1234.99!@*ASDasd*1234.99!@%&̂*_+-.'?[]{}|'~

FILTER_SANITIZE_NUMBER_FLOAT +- 123,456,789.8899 +-1234567898899

FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND +- 123,456,789.8899 +-123,456,7898899

FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION +- 123,456,789.8899 +-123456789.8899

FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND | FILTER_FLAG_ALLOW_FRACTION +- 123,456,789.8899 +-123,456,789.8899

FILTER_SANITIZE_NUMBER_INT +- 123,456,789 +-123456789

FILTER_SANITIZE_SPECIAL_CHARS ASDasd1234.99!@*ASDasd*1234.99!@%&*()_+-,./";'<>?:[]{}|'~

FILTER_SANITIZE_STRING ASDasd1234.99!@*ASDasd*1234.99!@%&̂*()_+-,./";'?:[]{}|'~

ASDasd1234.99!@#$%&̂*()_+-,./";'<>?:[]{}|'~ htmlspecialchars

©Jim Goudy

0.11 Defending Against Cross Site Scripting

Cross-Site Scripting (XSS) is one of the most common — and dangerous — web security flaws. At its core, XSS happens when untrusted content makes its way into a webpage without being properly handled, letting an attacker run malicious JavaScript in another user's browser.

The good news? A handful of defensive habits stop most XSS dead in its tracks. Below is a practical **XSS Defense Checklist** that explains not only what to do, but why it matters.

0.11.1 1. Escape Output (Context-Aware)

Escaping means converting potentially dangerous characters (`<`, `>`, `"`, `'`, `&`) into harmless equivalents before showing them in a web page. For example:

```
$name = "<script>alert(1)</script>";
echo htmlspecialchars($name, ENT_QUOTES, 'UTF -8');
```

This would display literally `<script>alert(1)</script>` in the browser instead of executing it. But here's the subtlety: **context matters.**

- **HTML body content:** Use `htmlspecialchars()` in PHP or `.textContent` in JS.
- **HTML attributes:** Make sure quotes are escaped.
- **JavaScript inline:** Encode data as JSON, not raw text.
- **CSS/URLs:** Encode values or, better, disallow dynamic injection entirely.

The key lesson: *Escape when outputting*, not when inputting. Input sanitization alone can't predict every possible context where data will appear.

0.11.2 2. Validate & Sanitize Input

Validation asks: *Is this data what I expect?*

- Last name $\rightarrow$ $letters, spaces, hyphensonly.$
- Email $\rightarrow$ $validemailformat.$

Sanitization cleans unwanted bits, but it should **never replace escaping.** Think of it as a helpful filter, not the main defense.

0.11.3 3. Use Content Security Policy (CSP)

A strong CSP header acts as a **seatbelt**:

```
Content -Security -Policy: default -src 'self'; script -src 'self'
```

Even if an attacker sneaks code in, CSP limits what can run. It's not a substitute for escaping, but it reduces the blast radius of mistakes.

0.11.4 4. Don't Trust JavaScript with HTML Injection

Tempting but dangerous:

- `innerHTML`
- `document.write()`
- `$('.element').html()`

Safer alternatives:

- `textContent`
- `createElement`
- Framework auto-escaping (React, Vue, Angular).

0.11.5 5. Use Frameworks Safely

Modern frameworks protect you — until you override them.

- React's `dangerouslySetInnerHTML`
- Vue's `v-html`

Both are escape hatches. Use them rarely, and only with fully trusted content.

0.11.6 6. Protect Your Forms

While not directly XSS, **CSRF tokens** stop attackers from tricking users into submitting malicious forms. Since XSS and CSRF often appear together, defense in depth is key.

0.11.7 7. Encode on the Way Out

Escaping at input time sounds convenient — but it's fragile. What if the data is later used in a different context (HTML vs. JS)?

Encoding **at output** ensures the escape matches the context every time. It's the difference between a one-size-fits-none approach and a tailored, future-proof defense.

0.11.8 8. Audit & Test

No defense is perfect without testing. Use:

- **Automated tools:** OWASP ZAP, Burp Suite.
- **Manual payloads:**

```
<script>alert(1)</script>
<img src=x onerror=alert(2)>
"><svg onload=alert(3)>
```

If any of these run, you've got work to do.

0.11.9 The Golden Rule

Treat all user input as hostile. Escape it when you output it. Do that, and you'll block 90% of XSS attempts before they ever start.

Escaping is your first line of defense. Validation, CSP, and safer coding practices are the reinforcements. Together, they form a layered security strategy that keeps both your applications and your users safe.

10/2025

0.11.10 XSS Examples

This short chapter demonstrates two common XSS scenarios and how to fix them:

1. **Reflected XSS** — user input (from the query string) is immediately written back into the page. The vulnerable example prints the raw query value; the fixed example escapes it with `htmlspecialchars` so any injected `<script>` becomes harmless text.
2. **Stored XSS (toy guestbook)** — a user-submitted message is stored in session and later rendered for all visitors. The vulnerable example renders the raw stored value, which will execute in the browser. The fixed example sanitizes stored values on output using a small `sanitize()` helper that performs `trim()`, `strip_tags()` and `htmlspecialchars()` (ENT_QUOTES, UTF-8).

Read the inline comments — I call out the vulnerable lines and explain the remediation. Use these pages in a classroom or lab to safely demonstrate how an attacker's `<script>` turns into harmless text once properly escaped.

`index.php` — home page with demo links

```
<?php
// index.php
// Simple landing page linking to the vulnerable and fixed demo pages.
//
// This file is purely HTML. Links include an example exploit payload to
// illustrate how a reflected XSS payload would look in the query string.
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF -8" />
    <meta name="viewport" content="width=device -width,initial -scale=1.0" />
    <title>XSS Home Demo</title>
    <style>
        .div1 {
            width: 50%;
            background -color: cornsilk;
            margin: 50px auto 0 auto;
            padding: 1rem;
            border -radius: 6px;
            font -family: system -ui, -apple -system, "Segoe UI", Roboto, "Helvetica Neue", Ar
        }
        a { display:block; margin:8px 0; }
    </style>
</head>
<body>
    <div class="div1">
        <h1>XSS Demos</h1>

        <! - -
          The query string contains a classic demo payload:
          ?q=<script>alert('Hacked')</script>
        - ->
```

```
        <a href="page1_reflect.php?q=<script>alert('Hacked')</script>">
            Page 1  Reflect (VULNERABLE)
        </a>

        <a href="page1_reflect_prevent.php?q=<script>alert('Hacked')</script>">
            Page 1  Reflect (Prevented)
        </a>

        <a href="stored_hacked.php">Stored  Guestbook (VULNERABLE)</a>
        <a href="stored_hacked_prevent.php">Stored  Guestbook (Prevented)</a>

        <a href="index.php">Home</a>
    </div>
</body>
</html>
```

`page1_reflect.php` — vulnerable reflected XSS (don't use in production)

```
<?php
// page1_reflect.php
// Demonstrates reflected XSS by echoing a query parameter directly.
// *** VULNERABLE: the raw $q value is injected into the HTML output. ***

// Use null coalescing so page still works without ?q= in the URL.
$q = $_GET['q'] ?? '';
?>
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf -8" />
    <title>Hacked (Reflect - Vulnerable)</title>
    <style>
        .div1 { width:50%; background:cornsilk; margin:50px auto; padding:1rem; }
    </style>
</head>
<body>
    <div class="div1">
        <h1>Search</h1>

        <! - - VULNERABLE LINE: raw echo of user input - ->
        <! - - If q contains <script>... it will run in the visitor's browser. - ->
        <p>You searched for: <?= $q ?></p> <! - - VULNERABLE - ->

        <br>
        <a href="index.php">Home</a>
    </div>
</body>
</html>
```

Teaching note: open `page1_reflect.php?q=<script>alert('Hacked')</script>` to see how an attacker-supplied `<script>` executes. The vulnerability comes from outputting unescaped input into an HTML context.

`page1_reflect_prevent.php` — reflected XSS fixed

```
<?php
// page1_reflect_prevent.php
// Fixed version of reflected XSS demo. Escape output with htmlspecialchars
// so any HTML tags supplied in the query string are rendered as text.

$q = $_GET['q'] ?? '';

// Use a small helper for clarity (context -aware escaping for HTML).
function e($s) {
    // ENT_QUOTES encodes both double and single quotes.
    // 'UTF -8' ensures proper encoding for multibyte characters.
    return htmlspecialchars((string)$s, ENT_QUOTES, 'UTF -8');
}
?>
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf -8" />
    <title>Hacked (Reflect - Prevented)</title>
    <style>.div1 { width:50%; background:cornsilk; margin:50px auto; padding:1rem; }</style>
</head>
<body>
    <div class="div1">
        <h1>Search</h1>

        <! - - SAFE OUTPUT: any < or > in $q are converted to &lt; &gt; - ->
        <p>You searched for: <?= e($q) ?></p> <! - - NOT VULNERABLE - ->

        <br>
        <a href="index.php">Home</a>
    </div>
</body>
</html>
```

Teaching note: `htmlspecialchars()` is the appropriate escape for HTML body content. If you output into other contexts (e.g., attribute, JS, CSS, URL) you must use a different encoding or a trusted library for context-aware encoding.

`stored_hacked.php` — vulnerable stored XSS guestbook

```
<?php
// stored_hacked.php
// Toy guestbook that demonstrates stored XSS by rendering session -stored messages
// without escaping them. DO NOT deploy this as -is.

session_start();
```

```
// Place a malicious demo message in session to simulate previously stored content.
$_SESSION['messages'] = ["<script>alert('hacked stored!')</script>"];

// Load messages for display.
$messages = $_SESSION['messages'] ?? [];

// If the user posts something, the code below intentionally follows your original
// request: keep the `if (!empty($_POST['msg']))` check exactly as provided.
if (!empty($_POST['msg'])) {
    // NOTE: this block is intentionally left minimal to match the original demo.
    // In the vulnerable demo we would append raw values to the messages array.
    // For clarity we simply reload messages here (no safe write).
    $messages = $_SESSION['messages'];
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf -8" />
    <title>Stored  Guestbook (Vulnerable)</title>
    <style>.div1 { width:50%; background:cornsilk; margin:50px auto; padding:1rem; }</style>
</head>
<body>
    <div class="div1">
        <h1>Guestbook</h1>

        <! - - Simple form to demonstrate posting - ->
        <form method="post" action="">
            <input name="msg" placeholder="Say hi" value="Hello">
            <button>POST</button>
        </form>

        <ul>
            <?php foreach ($messages as $m): ?>
                <! - - VULNERABLE: stored messages are printed raw, so a stored <script> will
                <li><?= (string) $m ?></li> <! - - VULNERABLE - ->
            <?php endforeach; ?>
        </ul>

        <br>
        <a href="index.php">Home</a>
    </div>
</body>
</html>
```

Teaching note: Stored XSS occurs when untrusted data is persisted (database, session, file) and later served to users without cleaning/escaping. The attack becomes more dangerous because many visitors might execute the injected script.

`stored_hacked_prevent.php` — stored XSS prevented (sanitizing on output)

```
<?php
```

```
// stored_hacked_prevent.php
// Prevents stored XSS by escaping/sanitizing messages before output.
// Uses the same ‘if (!empty($_POST[’msg’]))‘ check as requested.

session_start();

// Demo initial message that simulates previously stored malicious content.
$_SESSION[’messages’] = ["<script>alert(’hacked stored!’)</script>"];

// Retrieve messages array from session
$messages = $_SESSION[’messages’] ?? [];

// Keep original check as requested (no substitution)
if (!empty($_POST[’msg’])) {
    // In a real app we would append the submitted message to the session or DB.
    // For the classroom demo we show the flow but avoid writing raw content.
    $messages = $_SESSION[’messages’];
}

/**
 * sanitize()
 * - trim() removes leading/trailing whitespace
 * - strip_tags() removes HTML/PHP tags
 * - htmlspecialchars() escapes remaining special characters for HTML output
 *
 * This function is defensive and simple  it’s for demonstration and classroom use.
 * In production consider more context -aware measures (e.g., policies, CSP, output encoding l
 */
function sanitize($value)
{
    $value = trim((string)$value);                      // trim whitespace
    $value = strip_tags($value);                       // remove HTML tags
    $value = htmlspecialchars($value, ENT_QUOTES, ’UTF -8’); // escape for HTML context
    return $value;
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf -8" />
    <title>Stored  Guestbook (Prevented)</title>
    <style>.div1 { width:50%; background:cornsilk; margin:50px auto; padding:1rem; }</style>
</head>
<body>
    <div class="div1">
        <h1>Guestbook  Prevented</h1>

        <form method="post" action="">
            <input name="msg" placeholder="Say hi" value="Hello">
            <button>POST</button>
        </form>

        <ul>
```

```
            <?php foreach ($messages as $m): ?>
                <! - - SAFE: sanitize() neutralizes script tags and encodes special chars - ->
                <li><?= sanitize($m) ?></li> <! - - NOT VULNERABLE - ->
            <?php endforeach; ?>
        </ul>

        <br>
        <a href="index.php">Home</a>
    </div>
</body>
</html>
```

Teaching note: the safest approach is to **escape on output** for the context you are rendering into. Sanitizing on input can help (reduce stored attack surface), but **escaping at output** is the strongest, context-specific defense.

Observations

1. Visit `page1_reflect.php?q=<script>alert('XSS')</script>` and then `page1_reflect_prevent.php?q=<script>alert('XSS')</script>` — compare behavior.
2. Visit `stored_hacked.php` and observe the alert from the session-stored message. Then load `stored_hacked_prevent.php` to see the sanitized result.
3. Discuss: what happens if you put `onerror="..."` into an `<img>` tag inside a message? (Context matters — attribute vs body.)
4. Discuss Content Security Policy (CSP) as a further mitigation, and why it's complementary (not a substitute) to proper escaping.

10/2025

0.11.11 Content Security Policy

The Browser's Bodyguard

In the constant duel between security and convenience, **Cross-Site Scripting (XSS)** remains a classic ambush. Developers defend their web pages with careful *escaping* and *validation*, but even the best code occasionally slips. That's where **Content Security Policy (CSP)** steps in — not as a replacement, but as a **complementary layer of armor**.

---

1. What is Content Security Policy (CSP)?

CSP is an HTTP response header that acts as a browser-enforced whitelist. It tells the browser *exactly* which sources of scripts, styles, images, fonts, and frames are trusted — and blocks everything else.

In essence, CSP transforms the browser from a passive renderer into an active security guard.

Example:

```
Content -Security -Policy: default -src 'self'; script -src 'self' https://cdn.jsdelivr.net; o
```

This directive means:

- All content must come from the same origin (`'self'`).
- JavaScript may also load from **jsDelivr**.
- Plugins and embedded objects (like Flash or Java applets) are entirely forbidden.

The result: even if an attacker injects a malicious `<script>` tag, the browser refuses to execute it because it's not on the approved list.

---

2. Why CSP is Complementary, Not a Substitute

It's tempting to treat CSP as a magic fix for XSS — but that would be a mistake. **Escaping and validation** remain your *primary defense*; CSP simply limits the damage if something slips through.

Here's why they must work hand-in-hand:

1. **CSP restricts behavior, not content.** It tells the browser what to load, but it can't sanitize data. Escaping neutralizes harmful input before it ever reaches the browser.
2. **Misconfigurations happen.** A single misplaced `'unsafe-inline'` or forgotten domain can open the gates again. Proper escaping ensures that even missteps in policy remain survivable.
3. **Browser support varies.** Older or embedded browsers may ignore parts of CSP. Escaping works everywhere.
4. **Defense in depth.** Security thrives on redundancy. Escaping stops bad data; CSP catches anything that somehow sneaks through.

> Think of escaping as **locking the door**, and CSP as **posting a guard outside**.

---

3. Where and How to Apply a CSP Header

CSP is enforced by the browser but delivered by your server. You set it as an **HTTP response header**, usually through your web server configuration or application code.

Apache Example

```
Header set Content -Security -Policy "default -src 'self'; script -src 'self' https://cdn.jsde
```

Nginx Example

```
add_header Content -Security -Policy "default -src 'self'; script -src 'self' https://cdn.jsde
```

PHP Example

```
<?php
header("Content -Security -Policy: default -src 'self'; script -src 'self' https://cdn.jsdeliv
?>
```

(Place this before any HTML output so the header is sent correctly.)

---

4. The Meta Tag — A Limited Alternative

When you can't modify server headers (for instance, on a static host or learning environment), CSP can also be added via a `<meta>` tag:

```
<meta http -equiv="Content -Security -Policy" content="default -src 'self'; script -src 'self'
```

This works only for the current document and has limited enforcement power. It's fine for classroom demos or single pages, but in production, always use the **HTTP header** form.

---

5. Building a Policy in Practice

Start simple, test carefully, then tighten. CSP has a *report-only* mode that allows you to test your policy without breaking your site:

```
Content -Security -Policy -Report -Only: default -src 'self';
```

You can even log violations to a monitoring endpoint:

```
Content -Security -Policy: default -src 'self'; report -uri /csp -violation -report -endpoint/
```

This iterative approach lets you fine-tune your policy safely before enforcement.

---

6. The Belt and Suspenders Philosophy

Security is strongest when multiple defenses overlap.

- **Escaping**: Removes or neutralizes malicious input.
- **Validation**: Ensures only expected data enters the system.
- **CSP**: Stops execution or loading of untrusted resources.

Together, they form a resilient shield that protects your users, your data, and your reputation.

> Proper escaping is the *brakes*; CSP is the *seatbelt*. You need both to stay safe on the open web.

Summary Checklist

Layer	Purpose	Implementation
Escape Output	Prevent injection from rendering	`htmlspecialchars()` or equivalent
Validate Input	Reject invalid data early	Whitelist and sanitize
Use CSP Header	Restrict script and content sources	Set via HTTP header
Test with Report-Only	Monitor before enforcing	Log violations safely
Iterate Gradually	Refine over time	Add domains only when needed

---

Final Thought CSP isn't a silver bullet — but it's a brilliant ally. It turns the browser from a passive rendering engine into a **vigilant partner in security**. Use it thoughtfully, pair it with strong coding practices, and your web applications will stand firm against the most persistent adversaries of the modern web.

10/2025

0.12 Passwords

0.12.1 Key Ideas

- password_hash()
- password_verify()
- hashing
- salts

0.12.2 How PHP Passwords Work

`password_hash()` is a PHP function that creates a password hash using a strong one-way hashing algorithm The following algorithms are currently supported:

- `PASSWORD_DEFAULT`: Uses the bcrypt algorithm (default as of PHP 5.5.0). Note that this constant is designed to change over time as new and stronger algorithms are added to PHP. For that reason, the length of the result from using this identifier can change over time. Therefore, it is recommended to store the result in a database column that can expand beyond 60 characters (255 characters would be a good choice).
- `PASSWORD_BCRYPT`: Uses the CRYPT_BLOWFISH algorithm to create the hash. This will produce a standard crypt() compatible hash using the "2y2y" identifier. The result will always be a 60 character string, or false on failure.
- `PASSWORD_ARGON2I`: Uses the Argon2i hashing algorithm to create the hash. This algorithm is only available if PHP has been compiled with Argon2 support.
- `PASSWORD_ARGON2ID`: Uses the Argon2id hashing algorithm to create the hash. This algorithm is only available if PHP has been compiled with Argon2 support.

Here's an example of how to use `password_hash()`:

```
$password = "password123";
$hash = password_hash($password, PASSWORD_DEFAULT);
```

This code will generate a hash for `$password` using the default algorithm (`PASSWORD_DEFAULT`). The resulting hash can then be stored in a database for later use.

To verify a password, you can use the `password_verify()` function:

```
$password = "password123";
$hash = password_hash($password, PASSWORD_DEFAULT);

if (password_verify($password, $hash)) {
    echo "Password is valid!";
} else {
    echo "Invalid password.";
}
```

This code will verify that `$password` matches the hash generated by `password_hash()`. If they match, it will output "Password is valid!".

**Reference **

https://www.php.net/manual/en/function.password-hash.php

(AICHJIM)

How Password Salts Work

Password salting is a technique used to protect passwords stored in databases. It involves adding a random string of characters, called a salt, to the password before it is hashed. This makes it much more difficult for attackers to crack the passwords, even if they have access to the database.

Here's how password salting works:

1. **Salt generation:** A unique salt is generated for each password. This salt is typically a random string of characters that is at least 32 bytes long.
2. **Salt concatenation:** The salt is concatenated (joined) to the password. This creates a unique input for the hashing function.
3. **Hashing:** The combined salt and password are passed through a cryptographic hash function. This produces a unique hash value for each password.
4. **Storage:** The salt and hash value are stored together in the database. The original password is never stored.

> When a user attempts to log in, the salt is retrieved from the database and concatenated to the password entered by the user. This combination is then hashed, and the resulting hash value is compared to the stored hash value. If the two hash values match, the user is authenticated.

Salting provides several benefits for password security:

1. **Prevents Rainbow Table Attacks:** Rainbow tables are precomputed tables of hash values for common passwords. Attackers can use these tables to quickly crack passwords that have not been salted. However, salting makes rainbow table attacks ineffective because the salt changes the hash value for each password.
2. **Protects against Duplicate Passwords:** If two users have the same password, their hash values will be the same without salting. This makes it easier for attackers to identify common passwords. However, salting ensures that even if two users have the same password, their hash values will be different.
3. **Increases Password Complexity:** Salting effectively increases the complexity of passwords, making them more difficult to crack. This is because the salt adds additional entropy to the password, making it more difficult to guess or brute-force.

In summary, password salting is an essential security measure that helps protect passwords from being cracked. It is a simple but effective technique that should be used by all applications that store passwords.
(AIBJIM)

One Way Hashing

A one-way hashing algorithm, also known as a cryptographic hash function, is a mathematical function that takes an input of any length (a message or data) and produces a fixed-size output of random characters called a hash value or digest. The hash value is a unique representation of the input data, and it is practically impossible to reverse the process and obtain the original input from the hash value. This property makes one-way hashing algorithms crucial for data security and integrity verification.
Key characteristics of one-way hashing algorithms:

1. **Deterministic:** The same input always produces the same hash value.
2. **Irreversibility:** It is computationally infeasible to reverse the hash function and obtain the original input from the hash value.
3. **Collision Resistance:** It is extremely difficult to find two different inputs that produce the same hash value.
4. **Efficiency:** The hash function should be computationally efficient to calculate the hash value quickly.

Applications of one-way hashing algorithms:

1. **Password Protection:** Hashing is used to store passwords securely in databases. Instead of storing plain-text passwords, their hash values are stored, making it difficult for hackers to retrieve the original passwords even if they gain access to the database.
2. **Data Integrity:** Hash values are used to verify the integrity of data. By comparing the hash value of the original data with the hash value of the received data, one can ensure that the data has not been tampered with during transmission or storage.
3. **Digital Signatures:** Hash values are used in digital signatures to ensure the authenticity and non-repudiation of electronic documents.
4. **File Verification:** Hash values are used to verify the integrity of downloaded files, ensuring that the downloaded file is identical to the original file.
5. **Blockchain Technology:** Hashing is a fundamental component of blockchain technology, ensuring the immutability and security of the blockchain ledger.

Examples of widely used one-way hashing algorithms include MD5, SHA-1, SHA-256, and SHA-512.

(AIBJIM)

0.12.3 SQL Script

The key thing to look at there is the UNIQUE attribute for the username. Setting this to UNIQUE will cause the database to automatically check for duplicates. If it finds one, you can always trap the error and code an appropriate response.

```
- - SETUP DATABASE FOR FRESH INSTALL
DROP DATABASE IF EXISTS wp_newuser_demo;

CREATE DATABASE IF NOT EXISTS wp_newuser_demo;

USE  wp_newuser_demo;

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

 - - NOT IF YOU RUN THE CODE ABOVE,
 - - THIS STEP IS NOT NECESSARY
 - - THE CODE BELOW IF FOR REFFERENCE IF
 - - YOU CHOOSE TO KEEP AN EXISTING VERSION
 - - OF THE DATABASE

DROP TABLE IF EXISTS tbl_user;

 - - ALTERNATIVE IF YOU HAVE A TABLE
 - - BUT WISH TO REMOVE ALL THE DATA
 - - UNCOMMENT THE FOLLOWING

 - - TRUNCATE tbl_user;

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CREATE TABLE tbl_user
    (firstname VARCHAR(50),
```

```
    lastname VARCHAR(50),
    username VARCHAR (50) UNIQUE,
    password VARCHAR(255),
    user_ID  int(6)  AUTO_INCREMENT PRIMARY KEY);
```

0.12.4 Database Connection

Die() will cause the script to completely stop.

```
<?php

// dbconnect.php
//wp_newuser_demo
//tbl_user

// Turn off all error reporting
error_reporting(0);

try
{
    // connection information
    $host = "127.0.0.1";
    $dbname = "wp_newuser_demo";
    $user = "dev";
    $pw = "";

    // connect to database
 $pdo = new PDO("mysql:host=$host;dbname=$dbname",$user,$pw);
 $pdo ->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);

 //only for educational use to demonstration connection status
 $dbstatus = "Good database connection";

}
catch(PDOException $e)
{
    // get any error messages
 $dbstatus = "Database connection failed<br>".
     $e ->getMessage();

    // use in production
 //die();
}

// start session variables
SESSION_START();

// for lecture only
echo($dbstatus);
echo("<br><hr><br>");
```

0.12.5 Start Page

A simple page demonstrating new user and login options.

```
<! - - default.php - ->
<html>
<head>
<title>Home</title>
    <meta charset="UTF -8">
    <meta name="viewport" content="width=device -width, initial -scale=1.0">
    <link rel="stylesheet" href="myStyle.css">
</head>
<body>

<h1>Password Demo</h1>
<a href="NewUserRegistration.php">New User Registration</a><br><br>
<a href="login.php">Login</a><br><br>
<a href="memberpage.php">Member's Only</a><br><br>
<a href="logout.php">Logout</a><br><br>
</body>
</html>
```

0.12.6 New User Registration

```
<?php
// turn of error warnings - DO THIS LAST
error_reporting(0);

//NewUserRegistration.php
require("dbconnect.php");

// print_r($_POST);
echo("<br><br>");


function sanitize($value){

    // strip of any excess white spaces on the ends
    $value = trim($value);

    // get rid of any html or php tags
    $value = strip_tags($value);

    // convert special characters
    $value = htmlspecialchars($value,ENT_QUOTES,'UTF -8');

    // return the value
    return $value;

}


if(isset($_POST['username']))
{
 try
 {

 $pwd = $_POST['password'];
```

```
//sql statement
$sql_stmt = "INSERT INTO tbl_user (firstname,lastname,username,password)
       VALUES(:firstname,:lastname,:username,:password)";

//prepare
$sql_stmt = $pdo ->prepare($sql_stmt);

//sanitize
$firstname = sanitize($_POST['firstname']);
$lastname = sanitize($_POST['lastname']);
$username = sanitize($_POST['username']);
//option to clean the password

   // ********   PASSWORD *******
//hash the password
$password = password_hash($pwd, PASSWORD_DEFAULT);

   // ***************************

//bind param way
// $sql_stmt ->bindparam(":firstname",$firstname);
// $sql_stmt ->bindparam(":lastname",$lastname);
// $sql_stmt ->bindparam(":username",$username);
// $sql_stmt ->bindparam(":password",$password);

   // using array
   $param_array = array(
                    ':firstname' => $firstname,
                    ':lastname'=>$lastname,
                    ':username'=>$username,
                    ':password'=>$password);

//execute
$sql_stmt ->execute($param_array);
echo('<p>User was successfully entered</p>');

}
catch(PDOException $ee)
{

 echo($ee ->getMessage());
 echo("<br><br>");
 echo($ee ->getCode());

 if($ee ->getCode() == 23000)
 {
  //echo("Please select different username");
  $_SESSION['logerr_message'] = "Please select different username";
  echo('<script>alert("Plese use different username");</script>');
  header('location: NewUserRegistration.php');
       }
}
```

```
}
else
{
echo('
<!DOCTYPE html>
<html>
<head>
<title>New User</title>
    <meta charset="UTF -8">
    <meta name="viewport" content="width=device -width, initial -scale=1.0">
 <link rel="stylesheet" href="myStyle.css">

</head>
<body>

    <div class="div1">
        <h1>New User Registration</h1>


        <form method="POST" action="NewUserRegistration.php">
            <table border="1">
                <tr>
                    <td>First Name</td>
                    <td><input type="text" name="firstname" size="25"></td>
                </tr>
                <tr>
                    <td>Last Name</td>
                    <td><input type="text" name="lastname" size="25"></td>
                </tr>
                <tr>
                    <td>User Name</td>
                    <td><input type="text" name="username" size="25" required></td>
                </tr>
                <tr>
                    <td>Password</td>
                    <td><input type="password" name="password" size="25" required></td>
                </tr>
                <tr>
                    <td></td>
                    <td><input type="submit" name="submit"></td>
                </tr>
            </table>
        </form>
  <p><a href="default.php">Home</a></p>
    </div>
</body>
</html>
');
}
```

0.12.7 Login Page

On this page, note the **error_reporting ** function. Passing zero to it, will turn off error reporting.

```
<?php
// - - - - - - - - - - - - - - - - - - - -
//login.php
//https://www.php.net/manual/en/function.error -reporting.php

// Turn off all error reporting
error_reporting(0);


require('dbconnect.php');

if(isset($_POST['username']))
{

  $sql_login =  "SELECT username, password ".
      "FROM tbl_user ".
      "WHERE username = :username";

  //prepare
  $sql_login = $pdo ->prepare("$sql_login");

  //sanitize
  $username = filter_var($_POST['username'], FILTER_SANITIZE_STRING);

  //bind param
  $sql_login ->bindParam(":username",$username);

  //execute
  $sql_login ->execute();

  //get dataset / result
  $result = $sql_login ->fetch();

  if($result['password']== null)
  {
   echo("<br>Bad Username / or Password</br>");
  }
  else
  {
   //for your information - remove for production
   print_r($result['password']);
   echo('<br><br');

   //store the password
   $hash_db = $result['password'];

   //reverse the password process and compare
   if(password_verify($_POST['password'], $hash_db))
   {
    echo('<br><br>Password is valid');

    //This session variable will set as a flag to
    // let users log onto "Member / private" pages
```

```
    $_SESSION['LoginStatus'] = true;

    //Goto the members page
    header("location: memberpage.php");
   }
   else
   {
    //bad login
    $_SESSION['LoginStatus'] = false;

    // Note: you could set a session variable
    // and go to a customized error page
    echo('<br>Invalid Password<br>');
   }
  }
}
?>

<html>
<head>
<title>Login</title>
<meta charset="UTF -8">
<meta name="viewport" content="width=device -width, initial -scale=1.0">
<link rel="stylesheet" href="myStyle.css">
</head>
<body>
<div class="div1">
<form method="POST" action="Login.php">
<table border= "1">
 <tr>
  <td colspan="2">Login</td>
 </tr>
  <td>Username</td>
  <td><input type="text" name="username" size ="25" value="" require</td>
 </tr>
 <tr>
  <td>Password</td>
  <td><input type="password" name="password" size ="25" value="" require</td>
 </tr>
 <tr>
  <td></td>
  <td><input type="submit" name="submit" value="Enter"</td>
 </tr>
</table>
</form>

<a href="default.php">Home</a>
</div>
</body>
</html>
```

0.12.8 Logout

```
<?php
```

```
//logout.php

//Start Session
session_start();

// unset deletes the specified session variable
unset($_SESSION['LoginStatus']);

//redirect too the home page
header('location: default.php');
```

0.12.9 Member Page

```
<?php
// memberpage.php
//access the session variables
session_start();
print_r($_SESSION);

//*** Make sure to do this step
//*** It makes checks to see if the status flag is set to false to
//*** prevent unauthorised access to the page.
if(!isset($_SESSION['LoginStatus'])  || $_SESSION['LoginStatus']== false)
{
 header('Location:default.php');
}
?>
<html>
<head>
<title>Members</title>
    <meta charset="UTF -8">
    <meta name="viewport" content="width=device -width, initial -scale=1.0">
    <link rel="stylesheet" href="myStyle.css">
</head>
<body>
<div class="div1">
<h1>Members Only</h1>
<p>You are on the members only page.</p>
<p><a href="default.php">home</a> <a href="logout.php">Log Out</a></p>
</div>
</body>
</html>
```

0.13 Add Edit Delete

0.13.1 Database SQL

This SQL dump creates a small database named **wp_dogdb**, which is test database for storing information about dogs and U.S. states.

Here's what it does in essence:

1. **Creates the database** `wp_dogdb` (if it doesn't already exist).
2. **Defines two tables:**
   - `tbl_doginfo` — stores details about dogs and their owners, including:
     - `dogname`, `dogbreed`, `ownerfirstname`, `ownerlastname`, `city`, and `st` (state code).
     - Each record has a unique `dogid` (auto-incremented primary key).
   - `tbl_state` — a simple reference table listing all **U.S. state abbreviations**.
3. **Populates** `tbl_doginfo` with four sample dogs (all owned by "Joe Cool" from "Fluffy, AK").
4. **Populates** `tbl_state` with the full list of 50 U.S. state abbreviations.

```
- - phpMyAdmin SQL Dump
 - - version 4.7.0
 - - https://www.phpmyadmin.net/
 - -
 - - Host: 127.0.0.1
 - - Generation Time: Nov 02, 2017 at 04:53 PM
 - - Server version: 10.1.22 -MariaDB
 - - PHP Version: 7.1.4

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

 - -
 - - Database: ‘wp_dogdb‘
 - -
CREATE DATABASE IF NOT EXISTS ‘wp_dogdb‘ DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_c
USE ‘wp_dogdb‘;

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


 - -
 - - Table structure for table ‘tbl_doginfo‘
 - -

DROP TABLE IF EXISTS ‘tbl_doginfo‘;
CREATE TABLE IF NOT EXISTS ‘tbl_doginfo‘ (
  ‘dogname‘ varchar(20) NOT NULL,
```

```
  `dogbreed` varchar(20) NOT NULL,
  `ownerfirstname` varchar(20) NOT NULL,
  `ownerlastname` varchar(20) NOT NULL,
  `city` varchar(20) NOT NULL,
  `st` varchar(2) NOT NULL,
  `dogid` int(7) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`dogid`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;

 - -
 - - Dumping data for table `tbl_doginfo`
 - -

INSERT INTO `tbl_doginfo` (`dogname`, `dogbreed`, `ownerfirstname`, `ownerlastname`, `city`, `
('Barky', 'Spaniel', 'Joe', 'Cool', 'Fluffy', 'AK', 1),
('Barky 3', 'Spaniel', 'Joe', 'Cool', 'Fluffy', 'AK', 2),
('Barky 3', 'Spaniel', 'Joe', 'Cool', 'Fluffy', 'AK', 3),
('Barky 4', 'Spaniel', 'Joe', 'Cool', 'Fluffy', 'AK', 4);

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

 - -
 - - Table structure for table `tbl_state`
 - -

DROP TABLE IF EXISTS `tbl_state`;
CREATE TABLE IF NOT EXISTS `tbl_state` (
  `st` varchar(2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

 - -
 - - Dumping data for table `tbl_state`
 - -

INSERT INTO `tbl_state` (`st`) VALUES
('AL'),('AK'),('AZ'),('AR'),('CA'),('CO'),('CT'),('DE'),('FL'),('GA'),
('HI'),('ID'),('IL'),('IN'),('IA'),('KS'),('KY'),('LA'),('ME'),('MD'),
('MA'),('MI'),('MN'),('MS'),('MO'),('MT'),('NE'),('NV'),('NH'),('NJ'),
('NM'),('NY'),('NC'),('ND'),('OH'),('OK'),('OR'),('PA'),('RI'),('SC'),
('SD'),('TN'),('TX'),('UT'),('VT'),('VA'),('WA'),('WV'),('WI'),('WY');
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

0.13.2 dbconnect.php

This PHP script attempts to **connect to the `wp_dogdb` MySQL database** using PDO.

If the connection is successful, it sets a status message saying **"Good database connection."** If the connection fails, it catches the exception and stores an error message containing the failure reason.

Finally, it **starts a PHP session**—likely to maintain user or application state across pages.

```
<?php
```

```
try
{
    $pdo=new PDO("mysql:host=127.0.0.1;dbname=wp_dogdb",'root','');
    $pdo ->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);

    $dbstatus = "Good database connection";
}
 catch (PDOException $e)
 {
     $dbstatus = "Database connection failed <br>".
             $e ->getMessage();
 }
 session_start()

?>
```

0.13.3 index.php

This PHP and HTML code creates a simple web form for entering dog information into a database. It begins by including the `dbconnect.php` file to establish a connection with the `wp_dogdb` database and displays the current connection status. The script then retrieves a list of U.S. state abbreviations from the `tbl_state` table, sorting them alphabetically, and stores the results for use in the form. The webpage presents an input form where users can enter details about a dog—such as its name, breed, and the owner's first and last name, as well as the city and state. The state field is dynamically generated from the database query, ensuring the dropdown list reflects accurate state data. When the user submits the form, the entered information is sent via POST to `Inputdata_DisplayData.php` for further processing or display.

```
<!DOCTYPE html>
<html>
<head>
<title>Home Page</title>


<?php
 require 'dbconnect.php';

 echo $dbstatus;

    // used for building the select statement
 $sql_st = "SELECT * ".
   "FROM tbl_state ".
   "ORDER BY  st";

 //retreive the states and store them ind $dataset_st
 $dataset_st = $pdo ->query($sql_st);

?>

</head>
<body>

 <form method="POST" action = "Inputdata_DisplayData.php">
```

```
<table border="1">
 <tr>
  <td colspan="2">Dog Information</td>
 </tr>
 <tr>
  <td>Dog Name
           </td>
           <td>
           <input type="text" name="dogname" value="Barky">
           </td>
 </tr>
 <tr>
  <td>Dog Breed
           </td>
           <td>
           <input type="text" name="dogbreed" value="Mutt">
           </td>
 </tr>
 <tr>
  <td>Owner First Name
           </td>
           <td>
           <input type="text" name="ownerfirstname" value="Bubba">
           </td>
 </tr>
 <tr>
  <td>Owner Last Name
           </td>
           <td>
           <input type="text" name="ownerlastname" value="Smith">
           </td>
 </tr>
 <tr>
  <td>City
           </td>
           <td>
           <input type="text" name="city" value="Chicago">
           </td>
 </tr>
 <tr>
  <td>State
           </td>
           <td>
           <select name="st">
               <?php
                   while($row = $dataset_st ->fetch())
                   {
                   echo('<option value="'.$row['st'].'">'.$row['st'].'</option>');
                   }
               ?>
           </select>
           </td>
 </tr>
```

```
  <tr>
   <td>
            </td>
            <td><input type="submit" value="Enter">
            </td>
  </tr>
 </table>
 </form>
</body>
</html>
```

0.13.4 Inputdata_DisplayData.php

This PHP script, **Inputdata_DisplayData.php**, processes the dog information form submission from the previous page. It begins by including the database connection file `dbconnect.php`, then prints the submitted form data for debugging purposes. The script defines a `sanitize()` function to clean user input by trimming whitespace, removing HTML/PHP tags, and encoding special characters—ensuring the data is safe for database insertion.

Inside a `try` block, it prepares an SQL `INSERT` statement to add a new record into the `tbl_doginfo1` table, binding the sanitized form values (`dogname`, `dogbreed`, `ownerfirstname`, `ownerlastname`, `city`, and `st`) to named placeholders. After executing the insert command, the script displays a formatted confirmation message showing the details entered by the user. If any database or general errors occur, they are caught and displayed in a clear error message.

```
<?php
//Inputdata_DisplayData.php
require 'dbconnect.php';

print_r($_POST);
echo('<br><hr><br>');

function sanitize($value){

    // strip of any excess white spaces on the ends
    $value = trim($value);

    // get rid of any html or php tags
    $value = strip_tags($value);

    // convert special characters
    $value = htmlspecialchars($value,ENT_QUOTES,'UTF -8');

    // return the value
    return $value;

}

// alternative
function sanitize2($value){
    return htmlspecialchars(strip_tags(trim($value)),ENT_QUOTES,'UTF -8');
}

try{
    // write insert sql statement
```

```
$sql_insert =  "INSERT INTO tbl_doginfo1("
            ."dogname,"
            ."dogbreed,"
            ."ownerfirstname,"
            ."ownerlastname,"
            ."city,"
            ."st)"
            ."VALUES("
            .":dogname,"
            .":dogbreed,"
            .":ownerfirstname,"
            .":ownerlastname,"
            .":city,"
            .":st)";

// prepare the sql statement
$sql_insert = $pdo ->prepare($sql_insert);

// sanitize the information
$dogname = sanitize($_POST['dogname'] ?? '');
$dogbreed = sanitize($_POST['dogbreed'] ?? '');
$ownerfirstname = sanitize($_POST['ownerfirstname'] ?? '');
$ownerlastname = sanitize($_POST['ownerlastname'] ?? '');
$city = sanitize($_POST['city'] ?? '');
$st = sanitize($_POST['st'] ?? '');

//bind variable name to placeholders
$sql_insert ->bindparam(":dogname",$dogname);
$sql_insert ->bindparam(":dogbreed",$dogbreed);
$sql_insert ->bindparam(":ownerfirstname",$ownerfirstname);
$sql_insert ->bindparam(":ownerlastname",$ownerlastname);
$sql_insert ->bindparam(":city",$city);
$sql_insert ->bindparam(":st",$st);

//execute
$sql_insert ->execute();

// - -Alternative Binding Method using an array
/*
$sql_params=array(':dogname'=> $dogname,
                ':dogbreed' => $dogbreed,
                ':ownerfirstname' => $ownerfirstname ,
                ':ownerlastname' => $ownerlastname,
                ':city' => $city,
                ':st' => $st);

$sql_insert ->execute($sql_params);
 */



echo("
```

```
            <h2>Dog Infomation Confirmation</h2>
            Dog Name: $dogname<br>
            Dog Breed: $dogbreed<br>
            Owner First Name: $ownerfirstname<br>
            Owner Last Name: $ownerlastname<br>
            City: $city<br>
            State: $st<br>
        ");
}
catch(PDOException $err)
{
 echo('<h2>Error</h2>'.
  $err ->getMessage()
  );

}
catch(Exception $e)
{
 echo('<h2>Error</h2>'.
  $e ->getMessage()
  );
}

?>
```

0.13.5 Display_Edit.php

This PHP script displays and manages a **list of dogs** from the `tbl_doginfo` database table, allowing users to **edit or delete records** directly from a web interface.

It begins by including the `dbconnection.php` file to establish a database connection. When a form is submitted, the script checks the `action` value in the POST data:

- If the action is **"Delete"**, it sanitizes the `dogid` and executes a SQL `DELETE` command to remove the corresponding record.
- If the action is **"Edit"**, it stores the selected dog's ID in a session variable and redirects the user to `edit_data.php` to update that record.

After handling any actions, the script retrieves all existing dog records—showing each dog's name, breed, owner's name, city, and state—and dynamically generates an HTML table with **Edit** and **Delete** buttons for each entry.

In short: *This code powers a simple admin-style dashboard for managing dog records—complete with live database actions and a clean, table-based interface.*

```
<?php
require ('./dbconnection.php');

if (isset($_POST)) {

    if (!empty($_POST['action'])) {
        if ($_POST['action'] === 'Delete') {
            $dID = filter_var($_POST['dogid'], FILTER_SANITIZE_STRING);
            $sql_delete = "DELETE FROM tbl_doginfo WHERE dogid = " . $dID;
            $pdo ->exec($sql_delete);
        }

      if ($_POST['action'] === 'Edit')
```

```
      {
          //NOTE: you cannot send any output to the current page (Display_Edit.php
          //You must go directly to the new page.
          $_SESSION['dogEditID'] = filter_var($_POST['dogid'], FILTER_SANITIZE_NUMBER_INT);
          header("Location:edit_data.php");
      }
   }
}


$sql_selectEdit = "SELECT dogname, dogbreed, ownerfirstname, ownerlastname, city, st, dogid"
        . " FROM tbl_doginfo "
        . " ORDER BY dogname";

$result_edit = $pdo ->query($sql_selectEdit);
?>

<html>
    <head>
        <title></title>
    </head>
    <body>
<?php
include 'menu.php';
?>


        <table border="1">
            <thead>
                <tr>
                    <th>Dog Name</th>
                    <th>Dog Breed</th>
                    <th>First Name</th>
                    <th>Last Name</th>
                    <th>Owner First Name</th>
                    <th>Owner Last Name</th>
                    <th>Edit</th>
                </tr>
            </thead>
            <tbody>
<?php
while ($row = $result_edit ->fetch()) {
    echo('<tr>'
    . '<td>' . $row['dogname'] . "</td>"
    . "<td>" . $row['dogbreed'] . "</td>"
    . "<td>" . $row['ownerfirstname'] . "</td>"
    . "<td>" . $row['ownerlastname'] . "</td>"
    . "<td>" . $row['city'] . "</td>"
    . "<td>" . $row['st'] . "</td>"
    . '<td><form method="POST" action="Display_Edit.php"
      onsubmit="return confirm('."'".'Are U Sure'."')".'">
      <input type="hidden" name="dogid" value="'.$row['dogid'].'"/>'
    . '<input type="submit" value="Edit" name="action" />  '
```

```
    . '<input type="submit" value="Delete" name="action"/>'
    . '</form></td></tr>');
}
?>
            </tbody>
        </table>
    </body>
</html>
```

0.13.6 Edit_data.php

This PHP script provides an **edit interface for updating existing dog records** in the `tbl_doginfo` database.

It begins by including the database connection file and defining two `sanitize()` functions that clean user input by trimming whitespace, stripping HTML/PHP tags, and encoding special characters—ensuring the data is secure before being stored. If the user submits the form, the script prepares an SQL `UPDATE` statement with named placeholders and binds the sanitized input values (dog name, breed, owner info, city, and state) to those placeholders. It then executes the update and redirects the user back to the main display page.

If the form hasn't been submitted yet, the script retrieves the selected dog's record—based on a session-stored `dogEditID`—and queries the list of U.S. states from `tbl_state`. It then displays a pre-filled form that allows the user to modify the existing information, with the current state automatically selected.

In essence: *This page is the heart of the "edit dog" workflow—securely fetching, displaying, and updating canine records with clean, user-friendly precision.*

```
<?php
require ('./dbconnection.php');


function sanitize($value){

    // strip of any excess white spaces on the ends
    $value = trim($value);

    // get rid of any html or php tags
    $value = strip_tags($value);

    // convert special characters
    $value = htmlspecialchars($value,ENT_QUOTES,'UTF -8');

    // return the value
    return $value;

}

// alternative
function sanitize2($value){
    return htmlspecialchars(strip_tags(trim($value)),ENT_QUOTES,'UTF -8');
}


if (!empty($_POST['dogname'])) {
```

```
    //write the sql statement with placehonders
    $sql_edit = "UPDATE  tbl_doginfo "
            . "SET dogname = :dogname, "
            . "dogbreed = :dogbreed, "
            . "ownerfirstname = :ownerfirstname, "
            . "ownerlastname = :ownerlastname, "
            . "city = :city, "
            . "st = :st  "
            . "WHERE dogid = :dogid";

    //prepare the squel statement
    $sql_edit = $pdo ->prepare($sql_edit);


    //Sanitize Information
    $dogname = sanitize($_POST['dogname'] ?? '');
    $dogbreed = sanitize($_POST['dogbreed'] ?? '');
    $ownerfirstname = sanitize($_POST['ownerfirstname'] ?? '');
    $ownerlastname = sanitize($_POST['ownerlastname'] ?? '');
    $city = sanitize($_POST['city'] ?? '');
    $st = sanitize($_POST['st'] ?? '');


    //bind parameters
    $sql_edit ->bindparam(":dogname", $dogname);
    $sql_edit ->bindparam(":dogbreed", $dogbreed);
    $sql_edit ->bindparam(":ownerfirstname", $ownerfirstname);
    $sql_edit ->bindparam(":ownerlastname", $ownerlastname);
    $sql_edit ->bindparam(":city", $city);
    $sql_edit ->bindparam(":st", $st);
    $sql_edit ->bindparam(":dogid", $dogid);

 //input data
    $sqlh_edit ->execute();

 //open display data page
    header("Location: Display_Edit.php");
}


$sql_editData = "Select * "
        . " From tbl_doginfo"
        . " Where dogid="
        . $_SESSION['dogEditID'];

$result_editData = $pdo ->query($sql_editData);

$row_edit = $result_editData ->fetch();

// - - - -States
//Query to select states
$sql_st = "SELECT st " .
        "FROM tbl_state " .
```

```
        "Order by st";

//execute the select query
$result_st = $pdo ->query($sql_st);
?>

<html>
<head>
    <meta charset="UTF -8">
    <title>Edit Data</title>
</head>
<body>
    <?php
    include 'menu.php';
    ?>

    <h2>Edit Data</h2>
    <form method="POST" action="edit_data.php"
          onsubmit="return confirm('Are you sure you want to continue')">
        <table border="1">
            <thead>
                <tr>
                    <th colspan="2">Dog Information</th>
                </tr>
            </thead>
            <tbody>

                <tr>
                    <td>Dog Name</td>
                    <td><input type="text" name="dogname"
                               value="<?php echo $row_edit['dogname'] ?>" size="20" />      </
                </tr>
                <tr>
                    <td>Dog Breed</td>
                    <td><input type="text" name="dogbreed"
                               value="<?php echo $row_edit['dogbreed'] ?>" size="20" />      <
                </tr>
                <tr>
                    <td>Owner First Name</td>
                    <td><input type="text" name="ownerfirstname"
                           value="<?php echo $row_edit['ownerfirstname'] ?>" size="20" />
                </tr>
                <tr>
                    <td>Owner Last Name</td>
                    <td><input type="text" name="ownerlastname"
                               value="<?php echo $row_edit['ownerlastname'] ?>"
                size="20" /></td>
                </tr>
                <tr>
                    <td>City</td>
                    <td><input type="text" name="city"
                               value="<?php echo $row_edit['city'] ?>" size="20" /></td>
                </tr>
```

```
                <tr>
                    <td>ST</td>
                    <td><select name="st" size="1" width = 5>

                            <?php
                            while ($row = $result_st ->fetch()) {
                                if ($row['st'] === $row_edit['st']) {
                                    echo('<option value="'.$row['st']
                                         .'" selected>'.$row['st'].'</option>');
                                } else {
                                    echo('<option value="'.$row['st']
                                         . '">'.$row['st'].'</option>');
                                }
                            }
                            ?>
                        </select></td>
                </tr>
                <tr>
                    <td>Record Id: <?php echo $row_edit['dogid'] ?>
                        <input type="hidden" name="dogid" value="<?php
                        echo $row_edit['dogid'] ?> "/></td>
                    <td><input type="submit" value="Enter"/></td>
                </tr>
            </tbody>
        </table>
    </form>
</body>
</html>
```

0.13.7 menu.php

This short HTML snippet creates a simple **navigation menu** with two links: one leading to the **home page (`index.php`)** and another to the **data display page (`Display_Edit.php`)**. The links are separated by a small space and followed by line breaks for visual spacing.

In essence: *It's a minimalist navigation bar that helps users hop between the homepage and the dog data management page.*

```
<div>
    <a href="index.php">Home</a> 
    <a href="Display_Edit.php">Display Data</a><br><br>
</div>
```

10/2025

0.14 Saving Pictures In Database

0.14.1 Picture File Sizes

Upload sizes are controlled in the php.ini file

PHP allows shortcuts for byte values, including K (kilo), M (mega), and G (giga). PHP will do the conversions automatically if you use any of these. Be careful not to exceed the 32-bit signed integer limit (if you're using 32-bit versions) as it will cause your script to fail.

NOTE: The php.ini and the my.ini can be accessed through the XAMPP control panel.

Portions of the php.ini file

```
; Maximum size of POST data that PHP will accept.
; Its value may be 0 to disable the limit. It is ignored if POST data reading
; is disabled through enable_post_data_reading.
; https://php.net/post -max -size
post_max_size=40M

;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;

; Whether to allow HTTP file uploads.
; https://php.net/file -uploads
file_uploads=On

; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
; https://php.net/upload -tmp -dir
upload_tmp_dir="C:\xampp\tmp"

; Maximum allowed size for uploaded files.
; https://php.net/upload -max -filesize
upload_max_filesize=40M

; Maximum number of files that can be uploaded via a single request
max_file_uploads=20
```

And/Or

my.ini file for mysql

The error message "SQLSTATE[HY000]: General error: 2006 MySQL server has gone away" usually occurs when the MySQL server is not responding to the client's requests. This error can be caused by a variety of reasons, such as a low value of the max_allowed_packet variable, a low value of the wait_timeout variable, or insufficient memory on the server

To fix this error, you can try the following steps:

1. Increase the value of the max_allowed_packet variable in the my.cnf configuration file. You can set it to 16M or higher 1.
2. Increase the value of the wait_timeout variable in the my.cnf configuration file. You can set it to 28800 or higher 3.
3. Check the memory usage of the server. If the server is running out of memory, you can add a swap file to increase the available memory 1.

If none of these steps work, you can try to reconnect to the MySQL server and execute the query again 1. If the problem persists, you may need to contact your system administrator or MySQL support for further assistance.

```
# The MySQL server
default -character -set=utf8mb4
[mysqld]
port=3306
socket="C:/xampp/mysql/mysql.sock"
basedir="C:/xampp/mysql"
tmpdir="C:/xampp/tmp"
datadir="C:/xampp/mysql/data"
pid_file="mysql.pid"
# enable -named -pipe
key_buffer=16M
# **********Change to 16 or Greater ****************
max_allowed_packet=16M
# ************************************************
sort_buffer_size=512K
net_buffer_length=8K
read_buffer_size=256K
read_rnd_buffer_size=512K
myisam_sort_buffer_size=8M
log_error="mysql_error.log"
```

0.14.2 SQL Doggy Database

```
- - phpMyAdmin SQL Dump
 - - version 5.2.1
 - - https://www.phpmyadmin.net/
 - -
 - - Host: 127.0.0.1
 - - Generation Time: Nov 09, 2023 at 03:39 AM
 - - Server version: 10.4.28 -MariaDB
 - - PHP Version: 8.2.4

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

 - -
 - - Database: ‘wp_doggyembed‘
 - -
DROP DATABASE IF EXISTS ‘wp_doggyembed‘;
CREATE DATABASE IF NOT EXISTS ‘wp_doggyembed‘ DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_ge
USE ‘wp_doggyembed‘;

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
 - -
 - - Table structure for table `table1`
 - -

CREATE TABLE IF NOT EXISTS `table1` (
  `dogname` varchar(255) NOT NULL,
  `dogbreed` varchar(255) NOT NULL,
  `dogimage` longblob NOT NULL,
  `imagetype` varchar(255) NOT NULL,
  `imagesize` int(45) NOT NULL,
  `table1_pk` int(9) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`table1_pk`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

 - -
 - - Truncate table before insert `table1`
 - -

TRUNCATE TABLE `table1`;COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

0.14.3 dbconnect.php

```
<?php

try{

    //table1

    $host = "127.0.0.1:3308";
    $user = "root";
    $pw = "";

    // created a pdo object
    // in production change the user with appropriate privs
    $pdo = new PDO('mysql:host=127.0.0.1:3308;dbname=wp_doggyembed',$user,$pw);

    // lets setup some attributes for the object
    $pdo ->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);

    // only for educational purposes
    $dbstatus = "Good database connection";

    echo($dbstatus.'<br><hr><br>');

}
catch(PDOException $ep)
{
    $dbstatus = "Database connection failed<br>".
                    $ep ->getMessage();
```

```
    // for development purposes
    echo($dbstatus.'<br><hr><br>');

    die();
    // can also use exit
    // we don't page to load any further\
}
catch(Exception $e)
{
    $dbstatus = "Database connection failed<br>".
                    $e ->getMessage();

    echo($dbstatus.'<br><hr><br>');

    die();
}

session_start();

?>
```

0.14.4 index.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF -8">
    <meta name="viewport" content="width=device -width, initial -scale=1.0">
    <title>Home</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class = "content">
        <h1>Picture Embedding</h1>

        <p> <a href="doggyinput.php">Dog Picture Input</a>   
            <a href="getpix.php">Display Doggy Pictgures</a>    
            <a href="default.php">home</a>
        </p>

    </div>
</body>
</html>
```

0.14.5 doggyinput.php

The `enctype="multipart/form-data"` attribute in an HTML `<form>` specifies how form data should be encoded when it's submitted to the server. Here's a breakdown:

Context

- **Used in:** `<form>` element
- **Purpose:** Defines how form data is sent to the server.

Why use `multipart/form-data`?

It is necessary when a form includes file uploads, like images, documents, etc. It allows the form to send data in a way that supports sending both text and binary data.

How it works

When `enctype="multipart/form-data"` is set:

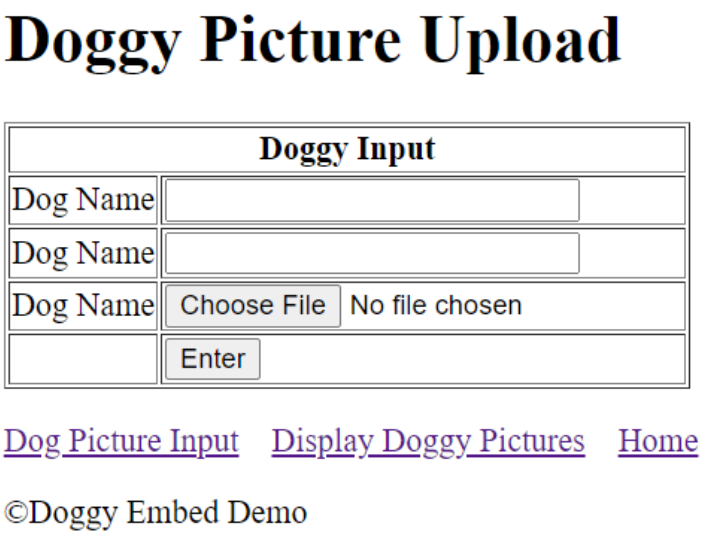
1. **Each form field's data** is sent as a separate part of the request body.
2. **File data** is encoded in binary, while other fields (e.g., text inputs) are encoded as plain text.
3. Each part is separated by a **boundary string** that the browser generates, making it easy for the server to parse the data correctly.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF -8">
    <meta name="viewport" content="width=device -width, initial -scale=1.0">
    <title>Input</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <h1>Doggy Picture Upload</h1>
    </header>
    <main>

<?php

require('dbconnect.php');

// for testing and demonstonly
// print_r($_POST);
// echo('<br>');
// print_r($_FILES);
// echo('<br><hr><br>');

if(isset($_POST['dogname']) && !empty($_POST['dogname']))
{


    // get file specifications
    $thedogfile = $_FILES['dogimage']['tmp_name'];
    $imagetype = mime_content_type($thedogfile);
    $imagesize = filesize($thedogfile);

    // Retrieve the file and encode it as base64
    // Base64 is a common form of handling data and it is used to ensure
    // that the handling of bits accros the network are correct.

    $dogimage = base64_encode(file_get_contents($_FILES['dogimage']['tmp_name']));

    // sql statement
```

```
    $sql = "INSERT INTO table1(dogname,dogbreed,dogimage,imagetype,imagesize) ".
            "VALUES(:dogname,:dogbreed,:dogimage,:imagetype,:imagesize)";

    // prepare sql statement
    $sqlobj = $pdo ->prepare($sql);

    // sanitize input
    $dogname = filter_var($_POST['dogname'],FILTER_SANITIZE_STRING);
    $dogbreed = filter_var($_POST['dogbreed'],FILTER_SANITIZE_STRING);

    // bind
    $sqlobj ->bindparam(':dogname',$dogname);
    $sqlobj ->bindparam(':dogbreed',$dogbreed);
    $sqlobj ->bindparam(':dogimage',$dogimage,PDO::PARAM_LOB);
    $sqlobj ->bindparam(':imagetype',$imagetype);
    $sqlobj ->bindparam(':imagesize',$imagesize);


    try{
        // execute the upload
        $uploadcheck = $sqlobj ->execute();
    }
    catch(PDOException $e)
    {
        $uploadcheck = false;
        echo("<br>Upload Failed: ".$e ->getMessage());
    }

    if($uploadcheck)
    {
        echo('
            <br><br>File Uploaded Successfully<br><br>
        ');
    }


}
else
{

    // *** NOTE enctype="multipart/form -data" ***
    // The enctype="multipart/form -data" attribute in an HTML <form>
    // is used when submitting forms that include file uploads.
    // It ensures that the form data is sent in separate parts,
    // allowing both text and binary data (like files) to be transmitted.
    // This encoding is necessary for handling file uploads,
    // as it allows the server to process each part correctly,
    // separating text fields from file data.
    // ****************************************************************


echo('
    <form action="doggyinput.php" method="POST" enctype="multipart/form -data">
```

```
        <table border="1">
            <tr>
                <th colspan=2>Doggy Input</th>
            </tr>
            <tr>
                <td>Dog Name</td>
                <td><input type="text" size="25" name="dogname"
                    value="barky" required> </td>
            </tr>
            <tr>
                <td>Dog Name</td>
                <td><input type="text" size="25" name="dogbreed"
                    value="mutt"> </td>
            </tr>
            <tr>
                <td>Dog Name</td>
                <td><input type="file"
                    name="dogimage"
                    accept=".jpg,.jpeg,.png"
                    required> </td>
            </tr>
            <tr>
                <td></td>
                <td><input type="submit" value="Enter"></td>
            </tr>
        </table>
        </form>
    ');
}  //end of else
?>
    <p><a href="doggyinput.php">Dog Picture Input</a>   
    <a href="getpix.php">Display Doggy Pictures</a>   
    <a href="index.php">Home</a>
</p></main>
<footer>&copy;Doggy Embed Demo</footer>
</body>
</html>');


<?php
/*
Upload sizes are controlled in the php.ini file

Items to check
; Maximum size of POST data that PHP will accept.
; Its value may be 0 to disable the limit. It is ignored if POST data reading
; is disabled through enable_post_data_reading.
; https://php.net/post -max -size
post_max_size=40M

Note:
PHP allows shortcuts for byte values, including K (kilo), M (mega) and G (giga).
PHP will do the conversions automatically if you use any of these. Be careful
```

```
not to exceed the 32 bit signed integer limit (if you're using 32bit versions)
as it will cause your script to fail.


;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;

; Whether to allow HTTP file uploads.
; https://php.net/file -uploads
file_uploads=On

; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
; https://php.net/upload -tmp -dir
upload_tmp_dir="C:\xampp\tmp"

; Maximum allowed size for uploaded files.
; https://php.net/upload -max -filesize
upload_max_filesize=40M

; Maximum number of files that can be uploaded via a single request
max_file_uploads=20


my.ini file
# The MySQL server
default -character -set=utf8mb4
[mysqld]
port=3306
socket="C:/xampp/mysql/mysql.sock"
basedir="C:/xampp/mysql"
tmpdir="C:/xampp/tmp"
datadir="C:/xampp/mysql/data"
pid_file="mysql.pid"
# enable -named -pipe
key_buffer=16M
# **********Change to 16 or Greater ****************
max_allowed_packet=16M
# ************************************************
sort_buffer_size=512K
net_buffer_length=8K
read_buffer_size=256K
read_rnd_buffer_size=512K
myisam_sort_buffer_size=8M
log_error="mysql_error.log"

*/

?>
```



0.14.6 getpix.php

```
<?php
    // database connection
    require('dbconnect.php');

    // SQL statement
    $sql = "SELECT * FROM table1";

    // Excecute query and store recordset
    $rs = $pdo ->query($sql);

?>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF -8">
<meta name="viewport" content="width=device -width, initial -scale=1.0">

<title>GetPix</title>
<link rel="stylesheet" href="style.css">
</head>

<body>
    <header>
        <h1>Doggy Pictures</h1>
        <p><a href="doggyinput.php">Dog Picture Input</a>   
        <a href="getpix.php">Display Doggy Pictures</a>   
        <a href="index.php">Home</a></p>
    </header>

    <main>
        <p>Here are the doggies</p>
```

```
        <?php
        while($row=$rs ->fetch())
        {
            echo('<div class="dogpix">');

            // retreive the image
            // build data url - example '.jpg'

            echo('<img src="data:'.$row['imagetype'].
                    ';base64,'.$row['dogimage'].'" width="300"><br>');
            echo($row['dogname'].'<br>'.$row['dogbreed']);

            echo('</div><br>');

        }
        ?>
        <p><a href="doggyinput.php">Dog Picture Input</a>   
        <a href="getpix.php">Display Doggy Pictures</a>   
        <a href="index.php">Home</a></p>
    </main>

    <footer>
        <p>&copy; 2023 Get Doggy Pix </p>
    </footer>
</body>
</html>
```

0.14.7 style.css

```
header, main, footer {
    width: 60%;
```

```
    margin -left: auto;
    margin -right:auto;
}

.dogpix {
    width: 300px;
    border: 3px solid darkblue;
    background -color: darkgrey;
    margin -left: auto;
    margin -right:auto;
    text -align: center;
    font -weight: bold;
}
```

0.15 Reading json

This code displays a table of Harry Potter spells on a webpage. Here's a breakdown:

- HTML Structure:
  - Defines a basic webpage with a title ("Spells") and a content area.
  - Styles the content area and a table to improve presentation.
- PHP Script:
  - Retrieves JSON data (currently from a specific URL) containing spell information.
  - Decodes the JSON data into an array for easier access.
  - Builds an HTML table with headers for "Spell Name," "Incantation," "Spell Type," and "Effect."
  - Loops through each spell in the array and displays its details in a table row.

```
<!DOCTYPE html>
<html>
<head>
  <title>Spells</title>
  <style>
    /* Style the table for better presentation */
    table {
      border -collapse: collapse; /* Remove borders between cells */
    }
    th {
      background -color: ivory; /* Set header background color */
      text -align: left; /* Align text left in headers */
    }
    tr:nth -child(even) {
      background -color: lightgrey; /* Alternate row background color */
    }
    .content {
      background -color: cornsilk; /* Set content area background color */
      margin: auto; /* Center the content horizontally */
      width: 65%; /* Set content area width to 50% */
    }
  </style>
</head>
<body>
<div class="content">
  <h1>Harry Potter Spells</h1>

  <?php

  // // Retrieve JSON data from a URL (replace with your actual URL)
  // $json = file_get_contents('url_here');
  // // Decode the JSON data into an object
  // $obj = json_decode($json);

  // Fetch JSON data from a specific URL (replace if needed)
  $url = 'https://flutter.locusnoesis.com/spellsinfo.php/';
  $json = file_get_contents($url);

  // Decode the JSON data into an array
  // Set second argument to true for associative array
```

```
  $spellsArr = json_decode($json, true);


  // Start building the HTML table
  echo('<table border="1">
          <tr>
            <th>Spell Name</th>  <th>Incantation</th>
            <th>Spell Type</th>
            <th>Effect</th>
          </tr>');

  // Loop through each spell in the array
  for ($r = 0; $r < count($spellsArr); $r++) {
    echo('<tr>');

    // Access the current spell object
    $currentSpell = $spellsArr[$r];

    // Loop through each property (key -value pair) of the current spell
    foreach ($currentSpell as $key => $value) {
      echo('<td>');
      echo($value);
      echo('</td>');
    }

    echo('</tr>');
  }

  echo('</table>');

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

Analysis:

- The code displays a table of Harry Potter spells.
- It retrieves data (currently from a specific URL) in JSON format and parses it into an array.
- The code loops through the spells and displays each spell's properties (name, incantation, type, effect) in a table row.
- The styling section defines the table's appearance and the content area's layout.

Improvements:

- The commented-out section shows how to replace the placeholder URL with your actual data source.
- The `json_decode` function's second argument is set to `true` to ensure an associative array is returned, making it easier to access properties by their names.
- The code assumes the JSON data structure is consistent. Error handling could be added to handle unexpected data formats.

0.16 Date Function

The PHP `date()` function is used to format a date and time according to a specified format.
##Syntax

```
date(format, timestamp)
```

0.16.1 Parameters

1. **`format`** (required)
   - The `format` parameter is a string that specifies how the outputted date should be formatted.
   - It uses a series of format characters, where each character represents a different aspect of the date/time.
   - Common format characters include:
     - `Y`: 4-digit year (e.g., 2024)
     - `y`: 2-digit year (e.g., 24)
     - `m`: 2-digit month (01 to 12)
     - `n`: Month without leading zeros (1 to 12)
     - `d`: 2-digit day of the month (01 to 31)
     - `j`: Day of the month without leading zeros (1 to 31)
     - `H`: 2-digit hour in 24-hour format (00 to 23)
     - `i`: 2-digit minutes (00 to 59)
     - `s`: 2-digit seconds (00 to 59)
     - `l`: Full day of the week (e.g., Monday)
     - `D`: Abbreviated day of the week (e.g., Mon)
     - `F`: Full month name (e.g., October)
     - `M`: Abbreviated month name (e.g., Oct)
2. **`timestamp`** (optional)
   - The `timestamp` is an optional integer parameter that specifies a Unix timestamp, representing the number of seconds since January 1, 1970.
   - If omitted, the function uses the current date and time.
   - Example: `date("Y-m-d", 1672531199)` would output the date corresponding to the given timestamp.

0.16.2 Examples

1. **Get the current date**

```
echo date("Y -m -d"); // Outputs: 2024 -10 -30
```

2. **Format a specific timestamp**

```
$timestamp = 1672531199;
echo date("Y -m -d H:i:s", $timestamp); // Outputs: 2023 -01 -01 00:59:59
```

The `date()` function is commonly used for formatting dates in various formats, making it versatile for generating date strings for display, logging, or processing purposes.

0.17 PHP + SQLite Shortcut Guide

A fast, practical reference for learning and teaching SQLite with PDO.

0.17.1 0. What Is SQLite?

SQLite is a **self-contained, serverless SQL database engine** that stores everything in **one single file**. There is *no* database server to install. There are *no* usernames, no passwords, no ports, no daemons, no configuration files.

If PHP can write to a folder, it can use SQLite.

SQLite is embedded in:

- PHP
- Python
- Android & iOS
- Firefox
- Windows & macOS
- Thousands of apps and devices

This makes SQLite:

- **Faster to set up** than MySQL
- **Simpler to teach** than PostgreSQL
- **Perfect for classroom exercises**
- **Rock-solid** for local or embedded data storage

0.17.2 Best Use Cases for SQLite

SQLite is your "pocket database"—small, steady, and always ready.

Ideal For:

- Teaching SQL and PDO
- Small to medium websites with modest traffic
- Desktop apps
- Mobile apps (Android/iOS)
- Prototyping and rapid development
- Apps that need offline local storage
- Research tools and student projects

Not Ideal For:

- High-traffic websites with heavy write operations
- Large multi-user applications
- Massive datasets (hundreds of GB)
- Systems requiring advanced permissions or replication

Use SQLite for **simplicity + reliability**. Use a server DB for **scalability + concurrency**.

0.17.3 0.5. How to Create a SQLite Database

SQLite database files are created **automatically**—as soon as PHP connects to a file that doesn't exist, SQLite creates it.

You can create databases in three main ways:

****Method 1 — Auto-Create Via PHP PDO ****

Just connect to a file. If it doesn't exist, SQLite generates it instantly:

```
$dbFile = __DIR__ . "/database.sqlite";

$db = new PDO("sqlite:" . $dbFile);
$db ->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Zero setup Creates the DB automatically Works on every OS Perfect for teaching

---

Method 2 — Create with sqlite3 Command Line

```
sqlite3 mydb.sqlite
```

In the SQLite shell:

```
CREATE TABLE test (id INTEGER);
.quit
```

Useful for:

- Inspecting the DB
- Running test queries
- Manually creating schemas

Method 3 — Create with a Graphical Tool (Beginner-Friendly)

Recommended GUIs:

- DB Browser for SQLite
- SQLiteStudio
- TablePlus

Steps:

1. Create a new database file
2. Add tables
3. Import/export data visually

Great for visual learners.

0.17.4 Best Location for Your Database File

```
project/
    db/
        database.sqlite
    public/
        index.php
```

Store the DB **outside the public folder** for security.

0.17.5 1. Connect to SQLite

```
$db = new PDO("sqlite:" . __DIR__ . "/database.sqlite");
$db ->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

0.17.6 2. Create a Table

```
$db ->exec("
    CREATE TABLE IF NOT EXISTS students (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        major TEXT NOT NULL,
        created_at TEXT NOT NULL
    )
");
```

0.17.7 3. Insert Data (CREATE)

```
$stmt = $db ->prepare("
    INSERT INTO students (name, major, created_at)
    VALUES (:name, :major, :created)
");
$stmt ->execute([
    ':name'    => $name,
    ':major'   => $major,
    ':created' => date('Y -m -d H:i:s')
]);
```

0.17.8 4. Read Data (SELECT)

```
$rows = $db ->query("
    SELECT * FROM students ORDER BY id DESC
") ->fetchAll(PDO::FETCH_ASSOC);
```

0.17.9 5. Search / Filtering

```
$stmt = $db ->prepare("
    SELECT * FROM students
    WHERE name LIKE :q OR major LIKE :q
");
$stmt ->execute([':q' => '%' . $search . '%']);
$rows = $stmt ->fetchAll(PDO::FETCH_ASSOC);
```

0.17.10 6. Update Data

```
$stmt = $db ->prepare("
    UPDATE students
    SET name = :name, major = :major
    WHERE id = :id
");
$stmt ->execute([
    ':name' => $newName,
    ':major' => $newMajor,
    ':id' => $id
]);
```

0.17.11 7. Delete Data

```
$stmt = $db ->prepare("DELETE FROM students WHERE id = :id");
$stmt ->execute([':id' => $id]);
```

0.17.12 8. Common SQLite Functions

Purpose	Function
Current timestamp	`datetime('now')`
Count rows	`COUNT(*)`
Max value	`MAX(column)`
Sort	`ORDER BY column DESC`
Limit	`LIMIT 10`

0.17.13 9. SQLite File Location Tips

```
$db = new PDO("sqlite:" . __DIR__ . "/db/mydb.sqlite");
```

0.17.14 10. Error Checking

```
try {
    // your PDO code
} catch (PDOException $e) {
    echo "Error: " . htmlspecialchars($e ->getMessage());
}
```

#3 **11. Security Must-Knows**

- Always use prepared statements
- Never embed variables directly in SQL strings
- Validate all user inputs
- Escape HTML output with `htmlspecialchars()`
- Keep SQLite files outside public web directories

0.17.15 12. Useful PDO Fetch Modes

```
PDO::FETCH_ASSOC
PDO::FETCH_NUM
PDO::FETCH_OBJ
PDO::FETCH_COLUMN
```

0.17.16 13. Check if a Table Exists

```
$result = $db ->query("
    SELECT name FROM sqlite_master
    WHERE type='table' AND name='students'
");
```

0.17.17 14. Drop a Table

```
$db ->exec("DROP TABLE IF EXISTS students");
```

#3 **15. List All Tables**

```
$tables = $db ->query("
    SELECT name FROM sqlite_master WHERE type='table'
") ->fetchAll(PDO::FETCH_COLUMN);
```

0.17.18 16. CRUD Workflow (Commit This to Memory)

```
1. Connect
2. Prepare
3. Bind
4. Execute
5. Fetch (optional)
```

0.17.19 17. Working with BLOBs (Images, PDFs, Binary Data)

SQLite supports BLOBs for storing binary data or Base64 strings.

Store a Raw BLOB

```
$data = file_get_contents("photo.jpg");

$stmt = $db ->prepare("
    INSERT INTO photos (title, image)
    VALUES (:t, :img)
");
$stmt ->bindValue(':t', "Sunset");
$stmt ->bindValue(':img', $data, PDO::PARAM_Lob);
$stmt ->execute();
```

Retrieve a BLOB

```
$stmt = $db ->prepare("SELECT image FROM photos WHERE id = :id");
$stmt ->execute([':id' => $id]);
$row = $stmt ->fetch(PDO::FETCH_ASSOC);

header("Content -Type: image/jpeg");
echo $row['image'];
```

Base64 Method (Great for Beginners)

```
$b64 = base64_encode(file_get_contents("photo.jpg"));
```

Display:

```
<img src="<?= $b64 ?>">
```

0.17.20 BLOB Use Cases

Teaching Thumbnails Small apps Offline tools
Large images Video files High-traffic apps
Production rule: **Store files on disk, store filenames in the DB.**

0.17.21 Search Filter - SQLITE

```
<?php
declare(strict_types=1);

$dbPath = __DIR__ . "/database.sqlite";
$pdo = new PDO("sqlite:" . $dbPath);
$pdo ->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Create table if it doesn't exist
$pdo ->exec("
    CREATE TABLE IF NOT EXISTS students (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        major TEXT NOT NULL,
        created_at TEXT NOT NULL
    )
");


// HANDLE FORM ACTIONS
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

// ADD new student
if (isset($_POST['add'])) {
    $name  = trim($_POST['name'] ?? '');
    $major = trim($_POST['major'] ?? '');

    if ($name !== '' && $major !== '') {
        $stmt = $pdo ->prepare("
            INSERT INTO students (name, major, created_at)
            VALUES (:n, :m, :c)
        ");
        $stmt ->execute([
            ':n' => $name,
            ':m' => $major,
            ':c' => date('Y -m -d H:i:s'),
        ]);
    }
}

// UPDATE existing student
if (isset($_POST['update'])) {
    $id    = (int)($_POST['id'] ?? 0);
    $name  = trim($_POST['edit_name'] ?? '');
    $major = trim($_POST['edit_major'] ?? '');

    if ($id > 0 && $name !== '' && $major !== '') {
        $stmt = $pdo ->prepare("
            UPDATE students
            SET name = :n, major = :m
            WHERE id = :id
        ");
        $stmt ->execute([
```

```
            ':n'  => $name,
            ':m'  => $major,
            ':id' => $id,
        ]);
    }
}

// DELETE student
if (isset($_POST['delete'])) {
    $id = (int)($_POST['id'] ?? 0);
    if ($id > 0) {
        $stmt = $pdo ->prepare("DELETE FROM students WHERE id = :id");
        $stmt ->execute([':id' => $id]);
    }
}


// SEARCH FILTER
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

$searchTerm = trim($_GET['q'] ?? '');

// If there is a search term, filter by name OR major
if ($searchTerm !== '') {
    $stmt = $pdo ->prepare("
        SELECT * FROM students
        WHERE name  LIKE :q
           OR major LIKE :q
        ORDER BY id DESC
    ");
    $stmt ->execute([':q' => '%' . $searchTerm . '%']);
    $rows = $stmt ->fetchAll(PDO::FETCH_ASSOC);
} else {
    // No search term: show all
    // $rows = $pdo ->query("
    //     SELECT * FROM students
    //     ORDER BY id DESC
    // ") ->fetchAll(PDO::FETCH_ASSOC);

    $sqlstmt = "SELECT * FROM students ORDER BY name";
    $stmt = $pdo ->query($sqlstmt);
    $rows = $stmt ->fetchAll(PDO::FETCH_ASSOC);
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF -8">
    <title>SQLite + PHP CRUD Demo</title>
    <style>
        body { font -family: Arial, sans -serif; max -width: 800px; margin: 20px auto; }
        form { margin -bottom: 15px; }
        .student -card { border: 1px solid #ccc; padding: 10px; margin -bottom: 10px; }
```

```
        .inline -form { display: inline -block; margin -right: 10px; }
        input[type="text"] { padding: 4px; margin -right: 4px; }
        button { padding: 4px 8px; }
    </style>
</head>
<body>
    <h2>SQLite + PHP CRUD Demo</h2>

    <! - - SEARCH FORM (GET) - ->
    <h3>Search Students</h3>
    <form method="GET">
        <input
            type="text"
            name="q"
            placeholder="Search by name or major"
            value="<?= htmlspecialchars($searchTerm) ?>"
        >
        <button type="submit">Search</button>
        <a href="search_filter.php"><button type="button">Clear</button></a>
    </form>

    <! - - ADD FORM - ->
    <h3>Add New Student</h3>
    <form method="POST">
        <input type="text" name="name" placeholder="Student Name" required>
        <input type="text" name="major" placeholder="Major" required>
        <button type="submit" name="add">Add Student</button>
    </form>

    <! - - STUDENT LIST - ->
    <h3>Students</h3>

    <?php if (empty($rows)): ?>
        <p><em>No students found.</em></p>
    <?php else: ?>
        <?php foreach ($rows as $r): ?>
            <div class="student -card">
                <p>
                    <strong>ID:</strong> <?= $r['id'] ?><br>
                    <strong>Name:</strong> <?= htmlspecialchars($r['name']) ?><br>
                    <strong>Major:</strong> <?= htmlspecialchars($r['major']) ?><br>
                    <small>Created at: <?= htmlspecialchars($r['created_at']) ?></small>
                </p>

                <! - - UPDATE FORM (inline) - ->
                <form method="POST" class="inline -form">
                    <input type="hidden" name="id" value="<?= $r['id'] ?>">
                    <input
                        type="text"
                        name="edit_name"
                        value="<?= htmlspecialchars($r['name']) ?>"
                        placeholder="Name"
                        required
```

```
                    >
                    <input
                        type="text"
                        name="edit_major"
                        value="<?= htmlspecialchars($r['major']) ?>"
                        placeholder="Major"
                        required
                    >
                    <button type="submit" name="update">Update</button>
                </form>

                <! - - DELETE FORM - ->
                <form method="POST" class="inline -form">
                    <input type="hidden" name="id" value="<?= $r['id'] ?>">
                    <button type="submit" name="delete" onclick="return confirm('Delete this s
                        Delete
                    </button>
                </form>
            </div>
        <?php endforeach; ?>
    <?php endif; ?>
</body>
</html>
```

YouTube Demo

```
<?php
// Simple YouTube video manager using PDO (PHP Data Objects) and SQLite.
// This script is a single -page app that can:
// - Add YouTube videos with a title, genre, and description
// - Store them in an SQLite database file
// - Show one selected video in an embedded player
// - List all saved videos and let you delete them

// Enforce strict typing for function parameters and return types.
// This helps catch some type -related mistakes early.
declare(strict_types=1);

// Build the path to the SQLite database file.
// __DIR__ is the directory where this PHP file lives.
$dbPath = __DIR__ . '/videos.sqlite';

// The DSN (Data Source Name) tells PDO which database driver to use and where it is.
// For SQLite, the format is "sqlite:/path/to/file".
$dsn = 'sqlite:' . $dbPath;

try {
    // Create a new PDO connection to the SQLite database.
    $pdo = new PDO($dsn);

    // Tell PDO to throw exceptions when errors happen.
    // This makes error handling easier.
    $pdo ->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    // If anything goes wrong with the connection, stop the script
    // and show a safe error message.
    die('Database connection failed: ' . htmlspecialchars($e ->getMessage()));
}

// Create the "videos" table if it does not already exist.
// This is a simple schema for our video manager.
$pdo ->exec(
    'CREATE TABLE IF NOT EXISTS videos (
        id INTEGER PRIMARY KEY AUTOINCREMENT,  - - Unique ID for each video
        title TEXT NOT NULL,                   - - Video title
        url TEXT NOT NULL,                     - - Original YouTube URL
        video_id TEXT NOT NULL,                - - Extracted YouTube video ID
        genre TEXT NOT NULL,                   - - Category/genre
        description TEXT NOT NULL,             - - Description written by the user
        created_at TEXT NOT NULL               - - When the entry was created (ISO timestamp)
    )'
);

// This will hold feedback messages for the user (success or error).
$message = '';

/**
 * Extract the 11 -character YouTube video ID from a URL.
```

```
 *
 * This function supports several common YouTube URL formats:
 *  - https://youtu.be/VIDEO_ID
 *  - https://www.youtube.com/watch?v=VIDEO_ID
 *  - https://www.youtube.com/embed/VIDEO_ID
 *  - https://www.youtube.com/shorts/VIDEO_ID
 *
 * If it can't find a valid ID, it returns null.
 */
function extractYouTubeId(string $url): ?string
{
    // Regular expressions to match different YouTube URL patterns.
    $patterns = [
        '#youtu\\.be/([\\w -]{11})#i',
        '#youtube\\.com/watch\\?v=([\\w -]{11})#i',
        '#youtube\\.com/embed/([\\w -]{11})#i',
        '#youtube\\.com/shorts/([\\w -]{11})#i',
    ];

    // Try each pattern until one matches.
    foreach ($patterns as $pattern) {
        if (preg_match($pattern, $url, $matches)) {
            // $matches[1] contains the captured video ID.
            return $matches[1];
        }
    }

    // If we get here, none of the simple patterns matched.
    // As a fallback, look at the query string (e.g., ?v=VIDEO_ID&t=123).
    $parts = parse_url($url);
    if (!empty($parts['query'])) {
        // Turn "key=value&key2=value2" into an associative array.
        parse_str($parts['query'], $query);
        // Check for a "v" parameter that looks like a valid video ID.
        if (!empty($query['v']) && preg_match('#^[\\w -]{11}$#', $query['v'])) {
            return $query['v'];
        }
    }

    // If no valid ID was found, return null.
    return null;
}

// Handle form submissions (POST requests).
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // "action" tells us what the user wants to do (add or delete).
    $action = $_POST['action'] ?? '';

    // Handle adding a new video.
    if ($action === 'add') {
        // Read and trim form inputs (remove extra spaces).
        $title = trim($_POST['title'] ?? '');
        $url = trim($_POST['url'] ?? '');
```

```
        $genre = trim($_POST['genre'] ?? '');
        $description = trim($_POST['description'] ?? '');

        // Basic validation: all fields are required.
        if ($title === '' || $url === '' || $genre === '' || $description === '') {
            $message = 'All fields are required.';
        } else {
            // Try to extract the YouTube video ID from the given URL.
            $videoId = extractYouTubeId($url);

            if ($videoId === null) {
                // If the URL doesn't look like a valid YouTube link.
                $message = 'Unable to determine YouTube video ID from the provided link.';
            } else {
                // Prepare an INSERT statement using named placeholders.
                // This helps prevent SQL injection and keeps code clean.
                $stmt = $pdo ->prepare(
                    'INSERT INTO videos (title, url, video_id, genre, description, created_at)
                     VALUES (:title, :url, :video_id, :genre, :description, :created_at)'
                );

                // Execute the prepared statement with actual values.
                $stmt ->execute([
                    ':title' => $title,
                    ':url' => $url,
                    ':video_id' => $videoId,
                    ':genre' => $genre,
                    ':description' => $description,
                    // Store the current time in a standard format (ISO 8601).
                    ':created_at' => (new DateTimeImmutable()) ->format(DateTimeInterface::ATO
                ]);

                $message = 'Video added successfully!';
            }
        }
    }
    // Handle deleting a video.
    elseif ($action === 'delete') {
        // Read the video ID from the form and cast it to an integer.
        $id = isset($_POST['id']) ? (int) $_POST['id'] : 0;

        if ($id > 0) {
            // Use a prepared statement to safely delete the row.
            $stmt = $pdo ->prepare('DELETE FROM videos WHERE id = :id');
            $stmt ->execute([':id' => $id]);
            $message = 'Video deleted.';
        }
    }
}

// Fetch all videos from the database, newest first.
$videosStmt = $pdo ->query('SELECT * FROM videos ORDER BY created_at DESC');
```

```
// fetchAll(PDO::FETCH_ASSOC) returns an array of associative arrays.
$videos = $videosStmt ->fetchAll(PDO::FETCH_ASSOC);

// Determine which video should be shown in the player.
// If the URL has ?watch=ID, use that. Otherwise, use the first video.
$selectedId = isset($_GET['watch']) ? (int) $_GET['watch'] : null;
$selectedVideo = null;

// Look through the list to find the selected video by ID.
if ($selectedId !== null) {
    foreach ($videos as $video) {
        if ((int) $video['id'] === $selectedId) {
            $selectedVideo = $video;
            break;
        }
    }
}

// If nothing is selected but there are videos, use the first one.
if ($selectedVideo === null && !empty($videos)) {
    $selectedVideo = $videos[0];
}

?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF -8">
    <! - - Make the page responsive on phones and tablets - ->
    <meta name="viewport" content="width=device -width, initial -scale=1.0">
    <title>YouTube Video Manager</title>
    <style>
        /* Basic page styling using CSS */
        body {
            font -family: Arial, sans -serif;
            margin: 0;
            padding: 0;
            background: #f3f4f6;
            color: #1f2933;
        }
        header {
            background: #1a73e8;
            color: white;
            padding: 1rem 2rem;
        }
        main {
            padding: 2rem;
            max -width: 1200px;
            margin: 0 auto;
        }
        .section {
            background: white;
            border -radius: 8px;
```

```
    padding: 1.5rem;
    margin -bottom: 1.5rem;
    box -shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
form .field {
    display: flex;
    flex -direction: column;
    margin -bottom: 1rem;
}
form label {
    font -weight: bold;
    margin -bottom: 0.5rem;
}
form input[type="text"],
form textarea {
    padding: 0.75rem;
    border: 1px solid #cbd5e1;
    border -radius: 4px;
    font -size: 1rem;
}
form textarea {
    resize: vertical;
    min -height: 100px;
}
form button {
    background: #1a73e8;
    color: white;
    border: none;
    padding: 0.75rem 1.5rem;
    font -size: 1rem;
    border -radius: 4px;
    cursor: pointer;
    transition: background 0.2s ease -in -out;
}
form button:hover {
    background: #1558b0;
}
.message {
    margin -bottom: 1rem;
    color: #0f5132;
}
.video -player iframe {
    width: 100%;
    height: 420px;
    border: none;
    border -radius: 8px;
}
table {
    width: 100%;
    border -collapse: collapse;
}
th, td {
    padding: 0.75rem;
```

```
            border -bottom: 1px solid #e2e8f0;
            text -align: left;
            vertical -align: top;
        }
        th {
            background: #f8fafc;
        }
        .actions {
            display: flex;
            gap: 0.5rem;
        }
        .actions a,
        .actions button {
            padding: 0.5rem 1rem;
            border -radius: 4px;
            text -decoration: none;
            font -size: 0.9rem;
        }
        .play -button {
            background: #22c55e;
            color: white;
        }
        .play -button:hover {
            background: #16a34a;
        }
        .delete -button {
            background: #ef4444;
            color: white;
            border: none;
            cursor: pointer;
        }
        .delete -button:hover {
            background: #dc2626;
        }
        .description {
            white -space: pre -wrap; /* Preserve line breaks in descriptions */
        }
        @media (max -width: 768px) {
            .actions {
                flex -direction: column;
            }
        }
    </style>
</head>
<body>
<header>
    <h1>YouTube Video Manager</h1>
</header>
<main>
    <! - - Section: Form to add a new video - ->
    <section class="section">
        <h2>Add a New Video</h2>
```

```
        <! - - Show any success or error message to the user - ->
        <?php if ($message !== ''): ?>
            <p class="message">
                <?php echo htmlspecialchars($message, ENT_QUOTES, 'UTF -8'); ?>
            </p>
        <?php endif; ?>

        <! - - "Add video" form - ->
        <form method="post">
            <! - - Hidden field tells PHP that this form is for "add" action - ->
            <input type="hidden" name="action" value="add">

            <div class="field">
                <label for="title">Title</label>
                <input type="text" id="title" name="title" required>
            </div>

            <div class="field">
                <label for="url">YouTube Link</label>
                <input type="text" id="url" name="url" required>
            </div>

            <div class="field">
                <label for="genre">Genre</label>
                <input type="text" id="genre" name="genre" required>
            </div>

            <div class="field">
                <label for="description">Description</label>
                <textarea id="description" name="description" required></textarea>
            </div>

            <button type="submit">Add Video</button>
        </form>
    </section>

    <! - - Section: Selected video player - ->
    <section class="section video -player">
        <h2>Selected Video</h2>

        <?php if ($selectedVideo): ?>
            <! - - Show the selected video's title - ->
            <h3>
                <?php echo htmlspecialchars($selectedVideo['title'], ENT_QUOTES, 'UTF -8'); ?>
            </h3>

            <! - - Embed the YouTube video using the extracted video_id - ->
            <iframe
                src="https://www.youtube.com/embed/<?php echo htmlspecialchars($selectedVideo[
                allow="accelerometer; autoplay; clipboard -write; encrypted -media; gyroscope;
                allowfullscreen>
            </iframe>
```

```
            <! - - Show the genre and description - ->
            <p>
                <strong>Genre:</strong>
                <?php echo htmlspecialchars($selectedVideo['genre'], ENT_QUOTES, 'UTF -8'); ?>
            </p>
            <p class="description">
                <?php
                // nl2br() converts line breaks to <br> tags for better formatting.
                echo nl2br(htmlspecialchars($selectedVideo['description'], ENT_QUOTES, 'UTF -8
                ?>
            </p>
        <?php else: ?>
            <! - - When no videos exist yet - ->
            <p>No videos available yet. Add one above to get started!</p>
        <?php endif; ?>
    </section>

    <! - - Section: Table listing all videos - ->
    <section class="section">
        <h2>All Videos</h2>

        <?php if (empty($videos)): ?>
            <p>No videos saved.</p>
        <?php else: ?>
            <table>
                <thead>
                <tr>
                    <th>Title</th>
                    <th>Genre</th>
                    <th>Description</th>
                    <th>Link</th>
                    <th>Actions</th>
                </tr>
                </thead>
                <tbody>
                <! - - Loop through all videos and show each in a table row - ->
                <?php foreach ($videos as $video): ?>
                    <tr>
                        <td>
                            <?php echo htmlspecialchars($video['title'], ENT_QUOTES, 'UTF -8')
                        </td>
                        <td>
                            <?php echo htmlspecialchars($video['genre'], ENT_QUOTES, 'UTF -8')
                        </td>
                        <td class="description">
                            <?php echo nl2br(htmlspecialchars($video['description'], ENT_QUOTE
                        </td>
                        <td>
                            <! - - Link to the original YouTube URL in a new tab - ->
                            <a href="<?php echo htmlspecialchars($video['url'], ENT_QUOTES, 'U
                               target="_blank" rel="noopener noreferrer">
                                Open
                            </a>
```

```
                        </td>
                        <td>
                            <div class="actions">
                                <! - - "Play" button just reloads the page with ?watch=ID - ->
                                <a class="play -button"
                                   href="?watch=<?php echo (int) $video['id']; ?>">
                                    Play
                                </a>

                                <! - - "Delete" form for this specific video - ->
                                <form method="post"
                                      onsubmit="return confirm('Are you sure you want to delet
                                    <input type="hidden" name="action" value="delete">
                                    <input type="hidden" name="id"
                                           value="<?php echo (int) $video['id']; ?>">
                                    <button type="submit" class="delete -button">
                                        Delete
                                    </button>
                                </form>
                            </div>
                        </td>
                    </tr>
                <?php endforeach; ?>
                </tbody>
            </table>
        <?php endif; ?>
    </section>
</main>
</body>
</html>
```