SRM Institute of Science and Technology

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Department of Computer Applications

Circular – 2023-24

B.Sc. CS 5th Sem

INTERNET PROGRAMMING (UCS23501J)- Lab Manual

Laboratory 1: Learning to Work with Linux Server

Title

Learning to Work with Linux Server

Aim

To familiarize students with the fundamental commands and environment of a Linux server, essential for web development and server management.

Procedure

1. **Accessing the Terminal:** Open a terminal window or connect to a Linux server via SSH.
2. **Basic Navigation:**
   - Use `pwd` to print the current working directory.
   - Use `ls` to list the contents of the current directory.
   - Use `ls -l` for a detailed list (long format).
   - Use `ls -a` to show all files, including hidden ones.
   - Use `cd <directory_name>` to change the current directory.
   - Use `cd ..` to go up one directory level.
   - Use `cd ~` to go to your home directory.
3. **Directory Management:**
   - Use `mkdir <directory_name>` to create a new directory.
   - Use `mkdir -p <path/to/new/directory>` to create nested directories.
   - Use `rmdir <directory_name>` to remove an empty directory.
4. **File Management:**
   - Use `touch <file_name>` to create an empty file.
   - Use `cp <source_file> <destination>` to copy a file.
   - Use `mv <source_file> <destination>` to move or rename a file.
   - Use `rm <file_name>` to remove a file.
   - Use `rm -r <directory_name>` to remove a directory and its contents (use with caution).
5. **Viewing File Content:**
   - Use `cat <file_name>` to display the entire content of a file.
   - Use `more <file_name>` to view file content page by page.
   - Use `less <file_name>` for more advanced viewing with scrolling.
6. **Getting Help:**
   - Use `man <command_name>` to view the manual page for a command.

Source Code

N/A (Command line operations)

Input

```
pwd
ls -l
mkdir my_directory
cd my_directory
touch my_file.txt
echo "Hello Linux" > another_file.txt
cat another_file.txt
rm my_file.txt
cd ..
rmdir my_directory
```

Expected Output

```
/home/user
total 0
drwxr-xr-x 2 user user 4096 May 21 10:00 my_directory
my_directory
Hello Linux
```

(Outputs will vary based on user's environment and specific commands executed)

Laboratory 2: Working with Files and Directory Commands

Title

Working with Files and Directory Commands

Aim

To gain proficiency in using various Linux commands for efficient management of files and directories.

Procedure

1. **Listing Files and Directories:**
   - `ls -l`: Long listing format, showing permissions, owner, group, size, and modification date.
   - `ls -a`: Lists all files, including hidden files (those starting with a dot).
   - `ls -lh`: Long listing with human-readable file sizes.
2. **Copying Files and Directories:**
   - `cp source_file destination_file`: Copies `source_file` to `destination_file`.
   - `cp -r source_directory destination_directory`: Recursively copies `source_directory` and its contents.
3. **Moving and Renaming Files/Directories:**
   - `mv old_name new_name`: Renames a file or directory.
   - `mv file_name destination_directory`: Moves a file to a new directory.
4. **Deleting Files and Directories:**
   - `rm file_name`: Deletes a file.
   - `rm -r directory_name`: Deletes a directory and its contents recursively.
   - `rm -f file_name`: Forces deletion without prompt.
   - `rm -rf directory_name`: Forces recursive deletion (use with extreme caution).
5. **Creating Directories:**
   - `mkdir directory_name`: Creates a new directory.
   - `mkdir -p path/to/new/directory`: Creates parent directories if they don't exist.
6. **Viewing File Content:**
   - `cat file_name`: Displays the entire content of a file to standard output.
   - `more file_name`: Displays file content one screen at a time, allowing forward navigation.
   - `less file_name`: Similar to `more`, but allows both forward and backward navigation.

Source Code

N/A (Command line operations)

Input

```
mkdir lab2_dir
cd lab2_dir
touch file1.txt file2.txt
echo "This is content for file1." > file1.txt
cp file1.txt file1_copy.txt
mv file2.txt renamed_file2.txt
```

```
ls -l
mkdir sub_dir
mv file1_copy.txt sub_dir/
ls -l sub_dir
cat file1.txt
rm renamed_file2.txt
cd ..
rm -r lab2_dir
```

Expected Output

```
total 8
-rw-r--r-- 1 user user 28 May 21 10:30 file1.txt
-rw-r--r-- 1 user user  0 May 21 10:30 file2.txt
drwxr-xr-x 2 user user 4096 May 21 10:30 sub_dir
total 4
-rw-r--r-- 1 user user 28 May 21 10:30 file1_copy.txt
This is content for file1.
```

(Outputs will vary based on user's environment and specific commands executed)

Laboratory 3: Working with File Commands, Creating and Modifying Files using Vi Editor

Title

File Commands and Vi Editor

Aim

To effectively use advanced file manipulation commands and master the creation and modification of files using the Vi text editor in a Linux environment.

Procedure

1. **Advanced File Commands:**
    - `head -n <number> <file_name>`: Displays the first `number` lines of a file.
    - `tail -n <number> <file_name>`: Displays the last `number` lines of a file.
    - `grep "pattern" <file_name>`: Searches for a specific pattern within a file.
    - `sort <file_name>`: Sorts the lines of a file alphabetically.
    - `wc -l <file_name>`: Counts the number of lines in a file.
    - `wc -w <file_name>`: Counts the number of words in a file.
    - `wc -c <file_name>`: Counts the number of characters in a file.
2. **Using Vi Editor:**
    - **Open/Create a file:** `vi <file_name>`
    - **Modes:**
        - **Command Mode:** Default mode when Vi opens. Used for navigation, deletion, copying, pasting, and saving.
        - **Insert Mode:** Used for typing text. Enter by pressing `i` (insert at cursor), `a` (append after cursor), `o` (open new line below).
        - **Last Line Mode (Ex Mode):** Used for saving, quitting, searching, and replacing. Enter by pressing `:`.
    - **Basic Vi Commands (in Command Mode):**
        - `i`: Insert text before the cursor.
        - `a`: Append text after the cursor.
        - `o`: Open a new line below the current line and enter insert mode.
        - `dd`: Delete the current line.
        - `yy`: Yank (copy) the current line.
        - `p`: Put (paste) the yanked/deleted content below the cursor.
        - `u`: Undo the last change.
        - `x`: Delete character under cursor.
    - **Saving and Quitting (in Last Line Mode):**
        - `:w`: Save the file.
        - `:q`: Quit Vi.
        - `:wq` or `ZZ`: Save and quit.
        - `:q!`: Quit without saving (discard changes).
    - **Searching and Replacing (in Last Line Mode):**
        - `:/<pattern>`: Search for `pattern` forward.
        - `s/<old>/<new>/g`: Replace all occurrences of `old` with `new` on the current line.
        - `:%s/<old>/<new>/g`: Replace all occurrences of `old` with `new` throughout the entire file.

Source Code

N/A (Command line operations and Vi editor commands)

Input

```
# Create a sample file
echo -e "apple\nbanana\norange\napple\ngrape" > fruits.txt

# Use file commands
head -n 2 fruits.txt
tail -n 1 fruits.txt
grep "apple" fruits.txt
sort fruits.txt
wc -l fruits.txt

# Vi Editor demonstration
vi demo.txt
# Inside Vi:
# i
# This is line 1.
# This is line 2.
# ESC
# :wq
```

Expected Output

```
apple
banana
grape
apple
apple
banana
grape
orange
5 fruits.txt
```

(After saving `demo.txt` in Vi, `cat demo.txt` should show its content)

```
This is line 1.
This is line 2.
```

Laboratory 4: Writing Simple PHP Programs

Title

Writing Simple PHP Programs

Aim

To introduce students to the basics of PHP programming, including script creation, variable declaration, and simple output.

Procedure

1. **Setup:** Ensure a web server (like Apache or Nginx) with PHP installed is running.
2. **Create a PHP file:** Use a text editor to create a file named `hello.php`.
3. **Basic "Hello, World!" Program:**
    - Add the standard PHP opening and closing tags.
    - Use the `echo` statement to print a string.
4. **Running the Program:**
    - Save the `hello.php` file in your web server's document root (e.g., `/var/www/html/` or `htdocs/`).
    - Open a web browser and navigate to `http://localhost/hello.php` (or your server's IP address).
5. **Variables and Data Types:**
    - Declare variables using the `$` prefix.
    - Assign different data types (string, integer, float).
    - Use `echo` to display variable values.
6. **Basic Arithmetic Operations:**
    - Perform addition, subtraction, multiplication, and division using variables.
    - Display the results.

Source Code

```
<?php
// hello.php
echo "<h1>Hello, World!</h1>";

// Variables and Data Types
$name = "Alice";
$age = 30;
$height = 1.75;
$isStudent = true;

echo "<p>Name: " . $name . "</p>";
echo "<p>Age: " . $age . "</p>";
echo "<p>Height: " . $height . "</p>";
echo "<p>Is Student: " . ($isStudent ? "Yes" : "No") . "</p>";

// Basic Arithmetic
$num1 = 10;
$num2 = 5;

$sum = $num1 + $num2;
$difference = $num1 - $num2;
$product = $num1 * $num2;
$quotient = $num1 / $num2;

echo "<h2>Arithmetic Operations</h2>";
```

```
echo "<p>Sum: " . $sum . "</p>";
echo "<p>Difference: " . $difference . "</p>";
echo "<p>Product: " . $product . "</p>";
echo "<p>Quotient: " . $quotient . "</p>";
?>
```

Input

N/A (The program directly defines values)

Expected Output

(Rendered in a web browser)

```
<h1>Hello, World!</h1>
<p>Name: Alice</p>
<p>Age: 30</p>
<p>Height: 1.75</p>
<p>Is Student: Yes</p>
<h2>Arithmetic Operations</h2>
<p>Sum: 15</p>
<p>Difference: 5</p>
<p>Product: 50</p>
<p>Quotient: 2</p>
```

Laboratory 5: Operators and Control Flow Statements

Title

PHP Operators and Control Flow Statements

Aim

To understand and implement various types of operators and fundamental control flow statements in PHP for decision-making and repetitive tasks.

Procedure

1. **Arithmetic Operators:** Demonstrate `+`, `-`, `*`, `/`, `%` (modulus), `**` (exponentiation).
2. **Assignment Operators:** Use `=`, `+=`, `-=`, `*=`, `/=`, `%=`.
3. **Comparison Operators:** Use `==` (equal), `===` (identical), `!=` (not equal), `!==` (not identical), `>`, `<`, `>=`, `<=`.
4. **Logical Operators:** Implement `&&` (AND), `||` (OR), `!` (NOT).
5. **Conditional Statements:**
    - **`if-else`:** Execute different blocks of code based on a condition.
    - **`if-elseif-else`:** Handle multiple conditions.
    - **`switch`:** Select one of many blocks of code to be executed.
6. **Looping Statements:**
    - **`for` loop:** Iterate a specific number of times.
    - **`while` loop:** Iterate as long as a condition is true.
    - **`do-while` loop:** Execute the block once, then repeat as long as a condition is true.
    - **`foreach` loop:** Iterate over elements of an array (covered in Lab 8, but can be introduced here simply).

Source Code

```
<?php
// operators_control_flow.php

echo "<h1>PHP Operators and Control Flow</h1>";

// --- Arithmetic Operators ---
echo "<h2>Arithmetic Operators</h2>";
$a = 20;
$b = 7;
echo "<p>a = $a, b = $b</p>";
echo "<p>Addition (a + b): " . ($a + $b) . "</p>";
echo "<p>Subtraction (a - b): " . ($a - $b) . "</p>";
echo "<p>Multiplication (a * b): " . ($a * $b) . "</p>";
echo "<p>Division (a / b): " . ($a / $b) . "</p>";
echo "<p>Modulus (a % b): " . ($a % $b) . "</p>";
echo "<p>Exponentiation (a ** 2): " . ($a ** 2) . "</p>";

// --- Assignment Operators ---
echo "<h2>Assignment Operators</h2>";
$x = 10;
echo "<p>Initial x: $x</p>";
$x += 5; // $x = $x + 5;
echo "<p>x after x += 5: $x</p>";
$x -= 3; // $x = $x - 3;
echo "<p>x after x -= 3: $x</p>";

// --- Comparison Operators ---
```

```
echo "<h2>Comparison Operators</h2>";
$val1 = 10;
$val2 = "10";
echo "<p>val1 = $val1, val2 = \"$val2\"</p>";
echo "<p>val1 == val2: " . (var_export($val1 == $val2, true)) . " (Value
comparison)</p>"; // true
echo "<p>val1 === val2: " . (var_export($val1 === $val2, true)) . " (Value
and type comparison)</p>"; // false
echo "<p>val1 > val2: " . (var_export($val1 > $val2, true)) . "</p>"; //
false

// --- Logical Operators ---
echo "<h2>Logical Operators</h2>";
$isSunny = true;
$isWarm = false;
echo "<p>Is Sunny: " . (var_export($isSunny, true)) . ", Is Warm: " .
(var_export($isWarm, true)) . "</p>";
echo "<p>Sunny AND Warm: " . (var_export($isSunny && $isWarm, true)) .
"</p>"; // false
echo "<p>Sunny OR Warm: " . (var_export($isSunny || $isWarm, true)) . "</p>";
// true
echo "<p>NOT Sunny: " . (var_export(!$isSunny, true)) . "</p>"; // false

// --- Conditional Statements (if-else, switch) ---
echo "<h2>Conditional Statements</h2>";
$score = 75;
if ($score >= 90) {
    echo "<p>Grade: A</p>";
} elseif ($score >= 80) {
    echo "<p>Grade: B</p>";
} elseif ($score >= 70) {
    echo "<p>Grade: C</p>";
} else {
    echo "<p>Grade: D</p>";
}

$day = "Wednesday";
switch ($day) {
    case "Monday":
        echo "<p>It's Monday!</p>";
        break;
    case "Wednesday":
        echo "<p>It's Hump Day!</p>";
        break;
    default:
        echo "<p>Some other day.</p>";
}

// --- Looping Statements (for, while, do-while) ---
echo "<h2>Looping Statements</h2>";

echo "<p>For loop (1 to 3):</p>";
for ($i = 1; $i <= 3; $i++) {
    echo "Iteration $i <br>";
}

echo "<p>While loop (count down from 3):</p>";
$count = 3;
while ($count > 0) {
    echo "Count: $count <br>";
    $count--;
}

echo "<p>Do-While loop (at least once):</p>";
$j = 0;
do {
```

```
    echo "Value of j: $j <br>";
    $j++;
} while ($j < 1);

?>
```

Input

N/A (Values are hardcoded in the script for demonstration)

Expected Output

(Rendered in a web browser)

```
<h1>PHP Operators and Control Flow</h1>
<h2>Arithmetic Operators</h2>
<p>a = 20, b = 7</p>
<p>Addition (a + b): 27</p>
<p>Subtraction (a - b): 13</p>
<p>Multiplication (a * b): 140</p>
<p>Division (a / b): 2.8571428571429</p>
<p>Modulus (a % b): 6</p>
<p>Exponentiation (a ** 2): 400</p>
<h2>Assignment Operators</h2>
<p>Initial x: 10</p>
<p>x after x += 5: 15</p>
<p>x after x -= 3: 12</p>
<h2>Comparison Operators</h2>
<p>val1 = 10, val2 = "10"</p>
<p>val1 == val2: true (Value comparison)</p>
<p>val1 === val2: false (Value and type comparison)</p>
<p>val1 > val2: false</p>
<h2>Logical Operators</h2>
<p>Is Sunny: true, Is Warm: false</p>
<p>Sunny AND Warm: false</p>
<p>Sunny OR Warm: true</p>
<p>NOT Sunny: false</p>
<h2>Conditional Statements</h2>
<p>Grade: C</p>
<p>It's Hump Day!</p>
<h2>Looping Statements</h2>
<p>For loop (1 to 3):</p>
Iteration 1 <br>Iteration 2 <br>Iteration 3 <br>
<p>While loop (count down from 3):</p>
Count: 3 <br>Count: 2 <br>Count: 1 <br>
<p>Do-While loop (at least once):</p>
Value of j: 0 <br>
```

Laboratory 6: Embedding PHP Script in HTML

Title

Embedding PHP Script in HTML

Aim

To demonstrate how to seamlessly integrate PHP code within HTML documents to generate dynamic web content.

Procedure

1. **Create an HTML file with `.php` extension:** Name the file `dynamic_page.php`. This is crucial for the web server to process the PHP code.
2. **Basic HTML Structure:** Start with a standard HTML5 boilerplate.
3. **Embed PHP Tags:** Place PHP code blocks (`<?php ... ?>`) anywhere within the HTML where dynamic content is needed.
4. **Display Dynamic Content:**
    - Use `echo` to output variables, expressions, or function results directly into the HTML.
    - Display the current date and time using PHP's `date()` function.
    - Use PHP to conditionally display HTML elements.
5. **Run the file:** Save `dynamic_page.php` in your web server's document root and access it via `http://localhost/dynamic_page.php`.

Source Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dynamic PHP Page</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-
color: #f4f4f4; }
        .container { background-color: #fff; padding: 20px; border-radius:
8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); }
        h1 { color: #333; }
        p { color: #666; }
        .highlight { color: blue; font-weight: bold; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Welcome to My Dynamic Page!</h1>

        <p>This page was generated dynamically using PHP.</p>

        <p>Current Server Time: <span class="highlight">
            <?php
            // Display the current date and time
            echo date("Y-m-d H:i:s");
            ?>
        </span></p>

        <?php
        // Define a variable in PHP
```

```
        $userName = "Guest";
        $isLoggedIn = true; // Change to false to see different output

        // Conditional display of content
        if ($isLoggedIn) {
            $userName = "John Doe"; // Assume user is logged in and name is
fetched
            echo "<p>Hello, <span class='highlight'>" . $userName . "</span>!
Glad to see you back.</p>";
        } else {
            echo "<p>Please <a href='#'>log in</a> to personalize your
experience.</p>";
        }

        // Loop to generate a list
        echo "<h2>My Favorite Fruits:</h2>";
        echo "<ul>";
        $fruits = ["Apple", "Banana", "Cherry", "Date"];
        foreach ($fruits as $fruit) {
            echo "<li>" . $fruit . "</li>";
        }
        echo "</ul>";
        ?>

        <p>Thank you for visiting!</p>
    </div>
</body>
</html>
```

Input

N/A (The PHP script generates content dynamically)

Expected Output

(Rendered in a web browser)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dynamic PHP Page</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-
color: #f4f4f4; }
        .container { background-color: #fff; padding: 20px; border-radius:
8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); }
        h1 { color: #333; }
        p { color: #666; }
        .highlight { color: blue; font-weight: bold; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Welcome to My Dynamic Page!</h1>

        <p>This page was generated dynamically using PHP.</p>

        <p>Current Server Time: <span class="highlight">2025-05-21
HH:MM:SS</span></p>
        <p>Hello, <span class='highlight'>John Doe</span>! Glad to see you
back.</p>
```

```
        <h2>My Favorite Fruits:</h2>
        <ul><li>Apple</li><li>Banana</li><li>Cherry</li><li>Date</li></ul>

        <p>Thank you for visiting!</p>
    </div>
</body>
</html>
```

Laboratory 7: Various Types of Parameters and Types of Functions in PHP

Title

PHP Functions and Parameters

Aim

To understand the concept of functions in PHP, including how to define them, pass different types of parameters (by value, by reference), and utilize various return types.

Procedure

1. **Defining Simple Functions:**
    - Create a function with no parameters and no return value.
    - Create a function with parameters and a return value.
2. **Parameter Types:**
    - **Pass by Value (Default):** Demonstrate how changes to parameters inside a function do not affect the original variable outside.
    - **Pass by Reference:** Use the `&` symbol to pass a variable by reference, allowing the function to modify the original variable.
    - **Default Parameter Values:** Define functions with optional parameters that have default values.
    - **Variable-length Argument Lists:** Briefly introduce `func_get_args()`, `func_num_args()`, `func_get_arg()` or the `...` operator for variadic functions (PHP 5.6+).
3. **Return Types:**
    - Functions returning simple data types (string, int, float, boolean).
    - Functions returning arrays or objects.
    - Type declarations for parameters and return values (PHP 7+).

Source Code

```
<?php
// functions_parameters.php

echo "<h1>PHP Functions and Parameters</h1>";

// --- 1. Function with no parameters and no return value ---
function greet() {
    echo "<p>Hello from the greet function!</p>";
}
greet(); // Call the function

// --- 2. Function with parameters and a return value ---
function addNumbers($num1, $num2) {
    return $num1 + $num2;
}
$sum = addNumbers(15, 25);
echo "<p>Sum of 15 and 25: " . $sum . "</p>";

// --- 3. Pass by Value (Default behavior) ---
function incrementByValue($value) {
    echo "<p>Inside incrementByValue (before): $value</p>";
    $value++;
    echo "<p>Inside incrementByValue (after): $value</p>";
}
$myNum = 10;
```

```
echo "<p>Outside (before call): $myNum</p>";
incrementByValue($myNum);
echo "<p>Outside (after call): $myNum</p>"; // Still 10, original variable
not changed

// --- 4. Pass by Reference ---
function incrementByReference(&$value) {
    echo "<p>Inside incrementByReference (before): $value</p>";
    $value++;
    echo "<p>Inside incrementByReference (after): $value</p>";
}
$anotherNum = 20;
echo "<p>Outside (before call): $anotherNum</p>";
incrementByReference($anotherNum);
echo "<p>Outside (after call): $anotherNum</p>"; // Now 21, original variable
changed

// --- 5. Default Parameter Values ---
function sayHello($name = "World") {
    echo "<p>Hello, $name!</p>";
}
sayHello(); // Uses default "World"
sayHello("PHP User"); // Uses provided name

// --- 6. Type Declarations (PHP 7+) ---
function multiply(int $a, int $b): int {
    return $a * $b;
}
echo "<p>Multiplication (5 * 4): " . multiply(5, 4) . "</p>";

// Example of variadic function (using ...)
function sumAll(...$numbers) {
    $total = 0;
    foreach ($numbers as $number) {
        $total += $number;
    }
    return $total;
}
echo "<p>Sum of 1, 2, 3, 4: " . sumAll(1, 2, 3, 4) . "</p>";
echo "<p>Sum of 10, 20: " . sumAll(10, 20) . "</p>";

?>
```

Input

N/A (Values are hardcoded in the script for demonstration)

Expected Output

(Rendered in a web browser)

```
<h1>PHP Functions and Parameters</h1>
<p>Hello from the greet function!</p>
<p>Sum of 15 and 25: 40</p>
<p>Outside (before call): 10</p>
<p>Inside incrementByValue (before): 10</p>
<p>Inside incrementByValue (after): 11</p>
<p>Outside (after call): 10</p>
<p>Outside (before call): 20</p>
<p>Inside incrementByReference (before): 20</p>
<p>Inside incrementByReference (after): 21</p>
<p>Outside (after call): 21</p>
<p>Hello, World!</p>
```

```
<p>Hello, PHP User!</p>
<p>Multiplication (5 * 4): 20</p>
<p>Sum of 1, 2, 3, 4: 10</p>
<p>Sum of 10, 20: 30</p>
```

Laboratory 8: Working with Strings and Arrays in PHP

Title

PHP Strings and Arrays

Aim

To explore and apply various built-in PHP functions for efficient manipulation of strings and different types of arrays.

Procedure

1. **String Manipulation:**
    - **Length:** `strlen()`
    - **Substring:** `substr()`
    - **Find Position:** `strpos()`
    - **Replace:** `str_replace()`
    - **Concatenation:** Using `.` operator.
    - **Case Conversion:** `strtolower()`, `strtoupper()`.
    - **Trim:** `trim()`, `ltrim()`, `rtrim()`.
2. **Array Types:**
    - **Indexed Arrays:** Create and access elements using numeric indices.
    - **Associative Arrays:** Create and access elements using named keys.
    - **Multidimensional Arrays:** Create arrays containing other arrays.
3. **Array Operations:**
    - **Count elements:** `count()`
    - **Add/Remove elements:** `array_push()`, `array_pop()`, `array_shift()`, `array_unshift()`.
    - **Sort arrays:** `sort()`, `rsort()` (indexed), `asort()`, `arsort()` (associative by value), `ksort()`, `krsort()` (associative by key).
    - **Iterate arrays:** Using `for` loop (indexed) and `foreach` loop (indexed and associative).
    - **Check existence:** `in_array()`, `array_key_exists()`.

Source Code

```
<?php
// strings_arrays.php

echo "<h1>PHP Strings and Arrays</h1>";

// --- String Manipulation ---
echo "<h2>String Manipulation</h2>";
$text = "  Hello, PHP World!  ";
echo "<p>Original text: '" . $text . "'</p>";

echo "<p>Length: " . strlen($text) . "</p>";
echo "<p>Trimmed: '" . trim($text) . "'</p>";
echo "<p>Uppercase: " . strtoupper($text) . "</p>";
echo "<p>Lowercase: " . strtolower($text) . "</p>";
echo "<p>Substring (from 9, 3 chars): " . substr($text, 9, 3) . "</p>"; //
"PHP"
echo "<p>Position of 'PHP': " . strpos($text, "PHP") . "</p>"; // 9
echo "<p>Replace 'World' with 'Universe': " . str_replace("World",
"Universe", $text) . "</p>";
```

```php
$greeting = "Good";
$time = "Morning";
$fullGreeting = $greeting . " " . $time . "!";
echo "<p>Concatenation: " . $fullGreeting . "</p>";

// --- Indexed Arrays ---
echo "<h2>Indexed Arrays</h2>";
$colors = ["Red", "Green", "Blue"];
echo "<p>Colors: " . $colors[0] . ", " . $colors[1] . ", " . $colors[2] .
"</p>";
echo "<p>Number of colors: " . count($colors) . "</p>";

array_push($colors, "Yellow"); // Add to end
echo "<p>After push: " . implode(", ", $colors) . "</p>";

$removedColor = array_pop($colors); // Remove from end
echo "<p>Removed: " . $removedColor . ", Remaining: " . implode(", ",
$colors) . "</p>";

sort($colors); // Sort alphabetically
echo "<p>Sorted colors: " . implode(", ", $colors) . "</p>";

echo "<p>Iterating with foreach:</p><ul>";
foreach ($colors as $color) {
    echo "<li>" . $color . "</li>";
}
echo "</ul>";

// --- Associative Arrays ---
echo "<h2>Associative Arrays</h2>";
$student = [
    "name" => "Alice",
    "age" => 22,
    "major" => "Computer Science"
];
echo "<p>Student Name: " . $student["name"] . "</p>";
echo "<p>Student Major: " . $student["major"] . "</p>";

$student["city"] = "New York"; // Add new element
echo "<p>Student with city: " . $student["name"] . " from " .
$student["city"] . "</p>";

echo "<p>Iterating with foreach:</p><ul>";
foreach ($student as $key => $value) {
    echo "<li>" . $key . ": " . $value . "</li>";
}
echo "</ul>";

asort($student); // Sort by value
echo "<p>Sorted by value:</p><pre>" . print_r($student, true) . "</pre>";

// --- Multidimensional Arrays ---
echo "<h2>Multidimensional Arrays</h2>";
$students = [
    ["name" => "Bob", "grade" => "A"],
    ["name" => "Charlie", "grade" => "B"]
];
echo "<p>First student's name: " . $students[0]["name"] . "</p>";
echo "<p>Second student's grade: " . $students[1]["grade"] . "</p>";

echo "<p>Iterating multidimensional array:</p><ul>";
foreach ($students as $s) {
    echo "<li>" . $s["name"] . " (Grade: " . $s["grade"] . ")</li>";
}
echo "</ul>";
```

```
?>
```

Input

N/A (Values are hardcoded in the script for demonstration)

Expected Output

(Rendered in a web browser)

```
<h1>PHP Strings and Arrays</h1>
<h2>String Manipulation</h2>
<p>Original text: '  Hello, PHP World!  '</p>
<p>Length: 21</p>
<p>Trimmed: 'Hello, PHP World!'</p>
<p>Uppercase:   HELLO, PHP WORLD!  </p>
<p>Lowercase:   hello, php world!  </p>
<p>Substring (from 9, 3 chars): PHP</p>
<p>Position of 'PHP': 9</p>
<p>Replace 'World' with 'Universe':   Hello, PHP Universe!  </p>
<p>Concatenation: Good Morning!</p>
<h2>Indexed Arrays</h2>
<p>Colors: Red, Green, Blue</p>
<p>Number of colors: 3</p>
<p>After push: Red, Green, Blue, Yellow</p>
<p>Removed: Yellow, Remaining: Red, Green, Blue</p>
<p>Sorted colors: Blue, Green, Red</p>
<p>Iterating with
foreach:</p><ul><li>Blue</li><li>Green</li><li>Red</li></ul>
<h2>Associative Arrays</h2>
<p>Student Name: Alice</p>
<p>Student Major: Computer Science</p>
<p>Student with city: Alice from New York</p>
<p>Iterating with foreach:</p><ul><li>name: Alice</li><li>age:
22</li><li>major: Computer Science</li><li>city: New York</li></ul>
<p>Sorted by value:</p><pre>Array
(
    [age] => 22
    [name] => Alice
    [major] => Computer Science
    [city] => New York
)
</pre>
<h2>Multidimensional Arrays</h2>
<p>First student's name: Bob</p>
<p>Second student's grade: B</p>
<p>Iterating multidimensional array:</p><ul><li>Bob (Grade:
A)</li><li>Charlie (Grade: B)</li></ul>
```

Laboratory 9: File and Directory Operations in PHP

Title

PHP File and Directory Operations

Aim

To learn how to perform common file system operations, such as creating, reading, writing, and deleting files, as well as managing directories using PHP functions.

Procedure

1. **File Creation and Writing:**
    - Use `fopen()` to open a file in write mode (`'w'`) or append mode (`'a'`).
    - Use `fwrite()` to write content to the file.
    - Use `fclose()` to close the file handle.
    - Alternatively, use `file_put_contents()` for simpler write operations.
2. **File Reading:**
    - Use `fopen()` to open a file in read mode (`'r'`).
    - Use `fread()` to read a specified number of bytes from the file.
    - Use `fgets()` to read a line at a time.
    - Use `file_get_contents()` to read the entire file into a string.
    - Use `readfile()` to read a file and write it to the output buffer.
3. **File Deletion:**
    - Use `unlink()` to delete a file.
4. **Checking File Properties:**
    - `file_exists()`: Checks if a file or directory exists.
    - `is_file()`: Checks if the path is a regular file.
    - `is_dir()`: Checks if the path is a directory.
    - `filesize()`: Gets the size of a file.
5. **Directory Management:**
    - `mkdir()`: Creates a new directory.
    - `rmdir()`: Removes an empty directory.
    - `scandir()`: Lists files and directories inside a specified path.
    - `is_readable()`, `is_writable()`: Check permissions.

Source Code

```
<?php
// file_directory_ops.php

echo "<h1>PHP File and Directory Operations</h1>";

$filename = "my_log.txt";
$dirname = "my_data";

// --- 1. Writing to a file ---
echo "<h2>File Writing</h2>";
try {
    // Using fopen, fwrite, fclose
    $fileHandle = fopen($filename, "w"); // 'w' truncates file to zero length
or creates it
    if ($fileHandle) {
        fwrite($fileHandle, "This is the first line.\n");
        fwrite($fileHandle, "This is the second line.\n");
```

```
        fclose($fileHandle);
        echo "<p>Content written to '$filename' using fopen/fwrite.</p>";
    } else {
        echo "<p style='color:red;'>Error: Could not open '$filename' for
writing.</p>";
    }

    // Using file_put_contents (simpler for quick writes)
    file_put_contents($filename, "Overwriting file with new content.\n");
    file_put_contents($filename, "Appending another line.\n", FILE_APPEND);
// Append mode
    echo "<p>Content overwritten and appended to '$filename' using
file_put_contents.</p>";

} catch (Exception $e) {
    echo "<p style='color:red;'>Error during file writing: " . $e-
>getMessage() . "</p>";
}

// --- 2. Reading from a file ---
echo "<h2>File Reading</h2>";
if (file_exists($filename)) {
    // Using file_get_contents
    $content = file_get_contents($filename);
    echo "<p>Content of '$filename' (file_get_contents):</p><pre>" .
htmlspecialchars($content) . "</pre>";

    // Using fopen, fread, fclose
    $fileHandle = fopen($filename, "r");
    if ($fileHandle) {
        echo "<p>Content of '$filename' (fread):</p><pre>";
        echo htmlspecialchars(fread($fileHandle, filesize($filename))); //
Read entire file
        fclose($fileHandle);
        echo "</pre>";
    }

    // Using readfile (directly outputs to browser)
    echo "<p>Content of '$filename' (readfile):</p><pre>";
    readfile($filename);
    echo "</pre>";

} else {
    echo "<p>File '$filename' does not exist for reading.</p>";
}

// --- 3. Directory Operations ---
echo "<h2>Directory Operations</h2>";
if (!is_dir($dirname)) {
    if (mkdir($dirname)) {
        echo "<p>Directory '$dirname' created successfully.</p>";
    } else {
        echo "<p style='color:red;'>Error: Could not create directory
'$dirname'.</p>";
    }
} else {
    echo "<p>Directory '$dirname' already exists.</p>";
}

// List contents of a directory
echo "<p>Contents of current directory:</p><ul>";
$currentDirContents = scandir('.'); // '.' refers to the current directory
foreach ($currentDirContents as $item) {
    echo "<li>" . htmlspecialchars($item) . "</li>";
}
echo "</ul>";
```

```
// List contents of the created directory
if (is_dir($dirname)) {
    echo "<p>Contents of '$dirname':</p><ul>";
    $dataDirContents = scandir($dirname);
    foreach ($dataDirContents as $item) {
        echo "<li>" . htmlspecialchars($item) . "</li>";
    }
    echo "</ul>";
}

// --- 4. Deleting a file ---
echo "<h2>File Deletion</h2>";
if (file_exists($filename)) {
    if (unlink($filename)) {
        echo "<p>File '$filename' deleted successfully.</p>";
    } else {
        echo "<p style='color:red;'>Error: Could not delete
'$filename'.</p>";
    }
} else {
    echo "<p>File '$filename' does not exist to delete.</p>";
}

// --- 5. Deleting a directory ---
echo "<h2>Directory Deletion</h2>";
// Note: rmdir() only removes empty directories.
// For non-empty directories, you'd need a recursive function.
if (is_dir($dirname)) {
    if (rmdir($dirname)) {
        echo "<p>Directory '$dirname' deleted successfully (if empty).</p>";
    } else {
        echo "<p style='color:red;'>Error: Could not delete directory
'$dirname' (it might not be empty).</p>";
    }
} else {
    echo "<p>Directory '$dirname' does not exist to delete.</p>";
}

?>
```

Input

N/A (The script creates and manipulates files/directories)

Expected Output

(Rendered in a web browser)

```
<h1>PHP File and Directory Operations</h1>
<h2>File Writing</h2>
<p>Content written to 'my_log.txt' using fopen/fwrite.</p>
<p>Content overwritten and appended to 'my_log.txt' using
file_put_contents.</p>
<h2>File Reading</h2>
<p>Content of 'my_log.txt' (file_get_contents):</p><pre>Overwriting file with
new content.
Appending another line.
</pre>
<p>Content of 'my_log.txt' (fread):</p><pre>Overwriting file with new
content.
Appending another line.
</pre>
```

```
<p>Content of 'my_log.txt' (readfile):</p><pre>Overwriting file with new
content.
Appending another line.
</pre>
<h2>Directory Operations</h2>
<p>Directory 'my_data' created successfully.</p>
<p>Contents of current
directory:</p><ul><li>.</li><li>..</li><li>file_directory_ops.php</li><li>my_
data</li></ul>
<p>Contents of 'my_data':</p><ul><li>.</li><li>..</li></ul>
<h2>File Deletion</h2>
<p>File 'my_log.txt' deleted successfully.</p>
<h2>Directory Deletion</h2>
<p>Directory 'my_data' deleted successfully (if empty).</p>
```

(Note: The exact output for directory contents may vary based on other files present in the server's directory.)

Laboratory 10: Exception Handling and Classes in PHP

Title

PHP Exception Handling and Classes

Aim

To implement robust error management using exception handling and to understand the fundamentals of Object-Oriented Programming (OOP) by defining and utilizing classes and objects in PHP.

Procedure

1. **Exception Handling:**
    - **`try-catch` block:** Enclose code that might throw an exception within a `try` block. Catch specific exceptions in `catch` blocks.
    - **`throw` keyword:** Manually throw an exception.
    - **`finally` block (PHP 5.5+):** Execute code regardless of whether an exception was thrown or caught.
    - Create a custom exception class by extending `Exception`.
2. **Classes and Objects:**
    - **Define a Class:** Use the `class` keyword to define a blueprint for objects.
    - **Properties (Attributes):** Declare variables within a class to hold data. Use access modifiers (`public`, `private`, `protected`).
    - **Methods (Behaviors):** Define functions within a class to perform actions.
    - **Constructor (`__construct()`):** A special method called automatically when an object is created.
    - **Creating Objects:** Use the `new` keyword to instantiate an object from a class.
    - **Accessing Properties and Methods:** Use the `->` operator.
    - **`$this` keyword:** Refers to the current object instance within a class method.

Source Code

```
<?php
// exceptions_classes.php

echo "<h1>PHP Exception Handling and Classes</h1>";

// --- 1. Exception Handling ---
echo "<h2>Exception Handling</h2>";

function divide($numerator, $denominator) {
    if ($denominator === 0) {
        throw new Exception("Division by zero is not allowed.");
    }
    return $numerator / $denominator;
}

// Example 1: Catching a standard Exception
echo "<h3>Standard Exception Handling</h3>";
try {
    echo "<p>Result of 10 / 2: " . divide(10, 2) . "</p>";
    echo "<p>Attempting 10 / 0...</p>";
    echo "<p>Result of 10 / 0: " . divide(10, 0) . "</p>"; // This line will
throw an exception
    echo "<p>This line will not be executed.</p>";
```

```
} catch (Exception $e) {
    echo "<p style='color:red;'>Caught exception: " . $e->getMessage() . "
(Code: " . $e->getCode() . ")</p>";
} finally {
    echo "<p>Finally block executed after division attempt.</p>";
}

// --- 2. Custom Exception Class ---
echo "<h3>Custom Exception Class</h3>";
class InvalidArgumentException extends Exception {
    public function __construct($message = "Invalid argument provided", $code
= 0, Throwable $previous = null) {
        parent::__construct($message, $code, $previous);
    }

    public function customErrorMessage() {
        return "Custom Error: " . $this->getMessage() . " on line " . $this-
>getLine() . " in " . $this->getFile();
    }
}

function processAge(int $age) {
    if ($age < 0 || $age > 120) {
        throw new InvalidArgumentException("Age must be between 0 and 120.");
    }
    echo "<p>Age processed: $age</p>";
}

try {
    processAge(30);
    processAge(150); // This will throw custom exception
} catch (InvalidArgumentException $e) {
    echo "<p style='color:red;'>Caught custom exception: " . $e-
>customErrorMessage() . "</p>";
} catch (Exception $e) { // Generic catch for any other exceptions
    echo "<p style='color:red;'>Caught generic exception: " . $e-
>getMessage() . "</p>";
}

// --- 3. Classes and Objects ---
echo "<h2>Classes and Objects</h2>";

class Car {
    // Properties
    public $brand;
    public $model;
    private $year; // Private property

    // Constructor
    public function __construct($brand, $model, $year) {
        $this->brand = $brand;
        $this->model = $model;
        $this->setYear($year); // Use setter for private property
        echo "<p>A new Car object ($brand $model) has been created.</p>";
    }

    // Public method
    public function getCarInfo() {
        return "This is a " . $this->year . " " . $this->brand . " " . $this-
>model . ".";
    }

    // Setter for private property
    public function setYear(int $year) {
        if ($year > 1900 && $year <= date("Y") + 1) {
            $this->year = $year;
```

```
        } else {
            echo "<p style='color:orange;'>Warning: Invalid year for car.
Setting to default 2000.</p>";
            $this->year = 2000;
        }
    }

    // Getter for private property
    public function getYear() {
        return $this->year;
    }
}

// Creating objects (instantiating the class)
$car1 = new Car("Toyota", "Camry", 2020);
$car2 = new Car("Honda", "Civic", 2018);
$car3 = new Car("Ford", "Mustang", 1850); // Invalid year

// Accessing properties and methods
echo "<p>" . $car1->getCarInfo() . "</p>";
echo "<p>" . $car2->getCarInfo() . "</p>";
echo "<p>" . $car3->getCarInfo() . "</p>";

// Attempt to access private property (will cause error/warning if
uncommented)
// echo $car1->year;

// Using getter/setter for private property
echo "<p>Car 1's year: " . $car1->getYear() . "</p>";
$car1->setYear(2022);
echo "<p>Car 1's year after update: " . $car1->getYear() . "</p>";

?>
```

Input

N/A (Values are hardcoded in the script for demonstration)

Expected Output

(Rendered in a web browser)

```
<h1>PHP Exception Handling and Classes</h1>
<h2>Exception Handling</h2>
<h3>Standard Exception Handling</h3>
<p>Result of 10 / 2: 5</p>
<p>Attempting 10 / 0...</p>
<p style='color:red;'>Caught exception: Division by zero is not allowed.
(Code: 0)</p>
<p>Finally block executed after division attempt.</p>
<h3>Custom Exception Class</h3>
<p>Age processed: 30</p>
<p style='color:red;'>Caught custom exception: Custom Error: Age must be
between 0 and 120. on line XX in /path/to/exceptions_classes.php</p>
<h2>Classes and Objects</h2>
<p>A new Car object (Toyota Camry) has been created.</p>
<p>A new Car object (Honda Civic) has been created.</p>
<p style='color:orange;'>Warning: Invalid year for car. Setting to default
2000.</p>
<p>A new Car object (Ford Mustang) has been created.</p>
<p>This is a 2020 Toyota Camry.</p>
<p>This is a 2018 Honda Civic.</p>
<p>This is a 2000 Ford Mustang.</p>
```

```
<p>Car 1's year: 2020</p>
<p>Car 1's year after update: 2022</p>
```

(Note: Line number and file path in custom exception message will vary based on your environment.)

Laboratory 11: Working with Database and Tables

Title

MySQL Database and Table Operations

Aim

To connect to a MySQL database server and perform fundamental database and table creation operations using SQL commands.

Procedure

1. **Install MySQL/MariaDB:** Ensure a MySQL or MariaDB server is installed and running on your system (or a remote server).
2. **Access MySQL Client:** Use a command-line client (like `mysql` or `mariadb`) or a GUI tool (like phpMyAdmin, MySQL Workbench).
3. **Login to MySQL:**
    - `mysql -u <username> -p` (then enter password)
4. **Create a Database:**
    - Use `CREATE DATABASE <database_name>;`
    - Verify creation: `SHOW DATABASES;`
5. **Select a Database:**
    - Use `USE <database_name>;` to specify the database for subsequent operations.
6. **Create a Table:**
    - Define table name, column names, data types, and constraints (e.g., `PRIMARY KEY`, `NOT NULL`, `AUTO_INCREMENT`).
    - Use `CREATE TABLE <table_name> (...)`
    - Verify table creation: `SHOW TABLES;`
    - View table structure: `DESCRIBE <table_name>;` or `DESC <table_name>;`
7. **Drop Database/Table (Optional, for cleanup):**
    - `DROP DATABASE <database_name>;`
    - `DROP TABLE <table_name>;`

Source Code

```
-- SQL commands for database and table operations

-- 1. Create a new database
CREATE DATABASE lab_db;

-- 2. Show all databases to verify creation
SHOW DATABASES;

-- 3. Select the newly created database to work with
USE lab_db;

-- 4. Create a table named 'students'
CREATE TABLE students (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    enrollment_date DATE DEFAULT CURRENT_DATE
);

-- 5. Show tables in the current database to verify creation
SHOW TABLES;
```

```
-- 6. Describe the structure of the 'students' table
DESCRIBE students;

-- Optional: Clean up (uncomment to drop if you want to re-run)
-- DROP TABLE students;
-- DROP DATABASE lab_db;
```

Input

The SQL commands listed in the "Source Code" section are the input to the MySQL client.

Expected Output

(Output from MySQL command-line client)

```
-- Output for CREATE DATABASE lab_db;
Query OK, 1 row affected (0.01 sec)

-- Output for SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| lab_db             |
+--------------------+
5 rows in set (0.00 sec)

-- Output for USE lab_db;
Database changed

-- Output for CREATE TABLE students (...);
Query OK, 0 rows affected (0.03 sec)

-- Output for SHOW TABLES;
+------------------+
| Tables_in_lab_db |
+------------------+
| students         |
+------------------+
1 row in set (0.00 sec)

-- Output for DESCRIBE students;
+------------------+--------------+------+-----+----------------+------------
-------+
| Field            | Type         | Null | Key | Default        | Extra
|
+------------------+--------------+------+-----+----------------+------------
-------+
| student_id       | int          | NO   | PRI | NULL           |
auto_increment    |
| first_name       | varchar(50)  | NO   |     | NULL           |
|
| last_name        | varchar(50)  | NO   |     | NULL           |
|
| email            | varchar(100) | YES  | UNI | NULL           |
|
| enrollment_date  | date         | YES  |     | current_date() |
|
```

```
+------------------+--------------+------+-----+----------------+------------
-------+
5 rows in set (0.00 sec)
```

Laboratory 12: Working with Queries, Joins and Sub Queries in MySQL

Title

MySQL Queries, Joins, and Subqueries

Aim

To master data retrieval from MySQL databases using various SQL query techniques, including filtering, sorting, aggregation, joining multiple tables, and utilizing subqueries.

Procedure

1. **Setup:** Ensure you have a MySQL database (e.g., `lab_db` from Lab 11) and populate it with sample data for at least two related tables (e.g., `students` and `courses`, or `employees` and `departments`).
2. **Basic Queries (`SELECT`, `WHERE`, `ORDER BY`, `LIMIT`):**
    - Select all columns from a table.
    - Select specific columns.
    - Filter rows using `WHERE` clause with various operators (`=`, `!=`, `>`, `<`, `LIKE`, `IN`, `BETWEEN`).
    - Sort results using `ORDER BY` (ASC/DESC).
    - Limit the number of results.
3. **Aggregate Functions (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`):**
    - Use aggregate functions to summarize data.
    - Group results using `GROUP BY`.
    - Filter grouped results using `HAVING`.
4. **Joins (`INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`):**
    - Combine rows from two or more tables based on a related column.
    - Demonstrate `INNER JOIN` (returns matching rows from both tables).
    - Demonstrate `LEFT JOIN` (returns all rows from the left table, and matching rows from the right).
    - Demonstrate `RIGHT JOIN` (returns all rows from the right table, and matching rows from the left).
5. **Subqueries:**
    - Use a `SELECT` statement within another SQL query.
    - Demonstrate subqueries in the `WHERE` clause (`IN`, `EXISTS`, comparison operators).
    - Demonstrate subqueries in the `FROM` clause (derived tables).
    - Demonstrate subqueries in the `SELECT` clause (scalar subqueries).

Source Code

```
-- SQL commands for queries, joins, and subqueries

-- Assume 'lab_db' database and 'students' table from Lab 11 exist.
-- Create another table 'courses' and 'enrollments' for join examples.

USE lab_db;

-- Create 'courses' table
CREATE TABLE courses (
    course_id INT AUTO_INCREMENT PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    credits INT NOT NULL
);
```

```sql
-- Create 'enrollments' table (many-to-many relationship)
CREATE TABLE enrollments (
    enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    grade VARCHAR(2),
    enrollment_date DATE DEFAULT CURRENT_DATE,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);

-- Insert sample data into 'students'
INSERT INTO students (first_name, last_name, email) VALUES
('Alice', 'Smith', 'alice@example.com'),
('Bob', 'Johnson', 'bob@example.com'),
('Charlie', 'Brown', 'charlie@example.com'),
('Diana', 'Prince', 'diana@example.com');

-- Insert sample data into 'courses'
INSERT INTO courses (course_name, credits) VALUES
('Introduction to Programming', 3),
('Database Management', 4),
('Web Development Basics', 3),
('Data Structures', 4);

-- Insert sample data into 'enrollments'
INSERT INTO enrollments (student_id, course_id, grade) VALUES
(1, 1, 'A'), -- Alice in Intro to Prog
(1, 2, 'B'), -- Alice in DB Mgmt
(2, 1, 'C'), -- Bob in Intro to Prog
(3, 3, 'A-'), -- Charlie in Web Dev
(4, 2, 'B+'), -- Diana in DB Mgmt
(4, 4, 'A'); -- Diana in Data Structures


-- --- Basic Queries ---
-- 1. Select all students
SELECT * FROM students;

-- 2. Select first name and email of students
SELECT first_name, email FROM students;

-- 3. Select students whose first name starts with 'A'
SELECT * FROM students WHERE first_name LIKE 'A%';

-- 4. Select students enrolled after '2025-01-01' (assuming current date)
SELECT * FROM students WHERE enrollment_date > '2025-01-01';

-- 5. Select students ordered by last name descending
SELECT * FROM students ORDER BY last_name DESC;

-- 6. Select top 2 students
SELECT * FROM students LIMIT 2;

-- --- Aggregate Functions ---
-- 7. Count total number of students
SELECT COUNT(*) AS total_students FROM students;

-- 8. Count enrollments per student
SELECT student_id, COUNT(*) AS num_courses_enrolled
FROM enrollments
GROUP BY student_id;

-- 9. Get average credits for all courses
SELECT AVG(credits) AS average_credits FROM courses;
```

```sql
-- 10. Get number of students per enrollment date, only for dates with more
than 1 student
SELECT enrollment_date, COUNT(student_id) AS num_students
FROM students
GROUP BY enrollment_date
HAVING COUNT(student_id) > 1;


-- --- Joins ---
-- 11. INNER JOIN: Get student names and the courses they are enrolled in
SELECT s.first_name, s.last_name, c.course_name
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id;

-- 12. LEFT JOIN: Get all students and their enrolled courses (if any)
SELECT s.first_name, s.last_name, c.course_name
FROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id
LEFT JOIN courses c ON e.course_id = c.course_id;

-- 13. RIGHT JOIN: Get all courses and the students enrolled in them (if any)
SELECT c.course_name, s.first_name, s.last_name
FROM students s
RIGHT JOIN enrollments e ON s.student_id = e.student_id
RIGHT JOIN courses c ON e.course_id = c.course_id;


-- --- Subqueries ---
-- 14. Subquery in WHERE clause (IN operator): Select students enrolled in
'Database Management'
SELECT first_name, last_name
FROM students
WHERE student_id IN (
    SELECT student_id
    FROM enrollments
    WHERE course_id = (SELECT course_id FROM courses WHERE course_name =
'Database Management')
);

-- 15. Subquery in FROM clause (Derived Table): Get students who have
enrolled in more than one course
SELECT s.first_name, s.last_name, enrolled_courses.num_courses
FROM students s
INNER JOIN (
    SELECT student_id, COUNT(course_id) AS num_courses
    FROM enrollments
    GROUP BY student_id
    HAVING COUNT(course_id) > 1
) AS enrolled_courses ON s.student_id = enrolled_courses.student_id;

-- 16. Subquery in SELECT clause (Scalar Subquery): Get student name and
their total number of courses
SELECT s.first_name, s.last_name,
       (SELECT COUNT(*) FROM enrollments e WHERE e.student_id = s.student_id)
AS total_courses
FROM students s;

-- Optional: Clean up created tables
-- DROP TABLE enrollments;
-- DROP TABLE courses;
```

Input

The SQL commands listed in the "Source Code" section are the input to the MySQL client.

Expected Output

(Outputs will vary based on the exact data and current date, but here's a general idea)

```
-- Output for SELECT * FROM students;
+------------+------------+-----------+-------------------+-----------------+
| student_id | first_name | last_name | email             | enrollment_date |
+------------+------------+-----------+-------------------+-----------------+
|          1 | Alice      | Smith     | alice@example.com | 2025-05-21      |
|          2 | Bob        | Johnson   | bob@example.com   | 2025-05-21      |
|          3 | Charlie    | Brown     | charlie@example.com | 2025-05-21
|
|          4 | Diana      | Prince    | diana@example.com | 2025-05-21      |
+------------+------------+-----------+-------------------+-----------------+

-- Output for INNER JOIN example (11)
+------------+-----------+---------------------------+
| first_name | last_name | course_name               |
+------------+-----------+---------------------------+
| Alice      | Smith     | Introduction to Programming |
| Alice      | Smith     | Database Management       |
| Bob        | Johnson   | Introduction to Programming |
| Charlie    | Brown     | Web Development Basics    |
| Diana      | Prince    | Database Management       |
| Diana      | Prince    | Data Structures           |
+------------+-----------+---------------------------+

-- Output for Subquery in WHERE clause (14)
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Alice      | Smith     |
| Diana      | Prince    |
+------------+-----------+

-- Output for Subquery in SELECT clause (16)
+------------+-----------+---------------+
| first_name | last_name | total_courses |
+------------+-----------+---------------+
| Alice      | Smith     |             2 |
| Bob        | Johnson   |             1 |
| Charlie    | Brown     |             1 |
| Diana      | Prince    |             2 |
+------------+-----------+---------------+
```

Laboratory 13: Manipulation of Cookies and Sessions using PHP

Title

Manipulation of Cookies and Sessions using PHP

Aim

To understand and implement client-side data storage using HTTP cookies and server-side data storage using PHP sessions for managing user-specific information across multiple page requests.

Procedure

1. **HTTP Cookies:**
    - **Setting a Cookie:** Use `setcookie()` function. Specify name, value, expiration time, path, domain, security (HTTPS), and httpOnly.
    - **Retrieving a Cookie:** Access cookie values using the `$_COOKIE` superglobal array.
    - **Deleting a Cookie:** Set the cookie's expiration time to a past date.
2. **PHP Sessions:**
    - **Starting a Session:** Use `session_start()` at the very beginning of each script that uses sessions.
    - **Storing Session Data:** Store data in the `$_SESSION` superglobal array.
    - **Retrieving Session Data:** Access session values from `$_SESSION`.
    - **Modifying Session Data:** Assign new values to `$_SESSION` variables.
    - **Unsetting Session Variables:** Use `unset()` to remove a specific session variable.
    - **Destroying a Session:** Use `session_destroy()` to completely end a session and remove all session data.

Source Code

```
<?php
// cookies_sessions.php

// --- Session Start (MUST be at the very top of the script) ---
session_start();

echo "<h1>PHP Cookies and Sessions</h1>";

// --- 1. Cookie Manipulation ---
echo "<h2>Cookie Manipulation</h2>";

$cookie_name = "user_preference";
$cookie_value = "dark_mode";
$cookie_expiry = time() + (86400 * 30); // 30 days

// Set a cookie
if (!isset($_COOKIE[$cookie_name])) {
    setcookie($cookie_name, $cookie_value, $cookie_expiry, "/"); // Path '/'
makes it available throughout the domain
    echo "<p>Cookie '$cookie_name' set to '$cookie_value' for 30 days.</p>";
} else {
    echo "<p>Cookie '$cookie_name' is already set to: " .
htmlspecialchars($_COOKIE[$cookie_name]) . "</p>";
}
```

```
// Retrieve a cookie
if (isset($_COOKIE[$cookie_name])) {
    echo "<p>Retrieved Cookie Value: " .
htmlspecialchars($_COOKIE[$cookie_name]) . "</p>";
} else {
    echo "<p>Cookie '$cookie_name' not found.</p>";
}

// Delete a cookie (uncomment to test deletion)
/*
if (isset($_COOKIE[$cookie_name])) {
    setcookie($cookie_name, "", time() - 3600, "/"); // Set expiry to a past
time
    echo "<p>Cookie '$cookie_name' deleted.</p>";
    // Note: To verify deletion, you might need to refresh the page twice
    // or check browser's cookie storage. $_COOKIE will still show it on the
current request.
}
*/

echo "<hr>";

// --- 2. Session Manipulation ---
echo "<h2>Session Manipulation</h2>";

// Store session data
if (!isset($_SESSION["views"])) {
    $_SESSION["views"] = 1;
    echo "<p>Session 'views' initialized to 1.</p>";
} else {
    $_SESSION["views"]++;
    echo "<p>Session 'views' incremented. Current views: " .
$_SESSION["views"] . "</p>";
}

if (!isset($_SESSION["username"])) {
    $_SESSION["username"] = "GuestUser";
    echo "<p>Session 'username' set to 'GuestUser'.</p>";
} else {
    echo "<p>Session 'username' is: " .
htmlspecialchars($_SESSION["username"]) . "</p>";
}

// Modify session data
$_SESSION["last_activity"] = time();
echo "<p>Session 'last_activity' updated to: " . date("H:i:s",
$_SESSION["last_activity"]) . "</p>";

// Unset a specific session variable (uncomment to test unsetting)
/*
if (isset($_SESSION["username"])) {
    unset($_SESSION["username"]);
    echo "<p>Session 'username' unset.</p>";
}
*/

// Destroy the entire session (uncomment to test destruction)
/*
echo "<p>Destroying session...</p>";
session_unset();     // Unset all session variables
session_destroy();   // Destroy the session
echo "<p>Session destroyed. Refresh to see 'views' reset.</p>";
*/

echo "<p><a href='cookies_sessions.php'>Refresh Page to see Session/Cookie
changes</a></p>";
```

```
?>
```

Input

N/A (The script directly manipulates cookies and sessions. User interaction is primarily via page refresh.)

Expected Output

(Rendered in a web browser. Output changes with each page refresh.)

First Load:

```
<h1>PHP Cookies and Sessions</h1>
<h2>Cookie Manipulation</h2>
<p>Cookie 'user_preference' set to 'dark_mode' for 30 days.</p>
<p>Retrieved Cookie Value: dark_mode</p>
<hr>
<h2>Session Manipulation</h2>
<p>Session 'views' initialized to 1.</p>
<p>Session 'username' set to 'GuestUser'.</p>
<p>Session 'last_activity' updated to: HH:MM:SS</p>
<p><a href='cookies_sessions.php'>Refresh Page to see Session/Cookie
changes</a></p>
```

Second Load (and subsequent loads without clearing cookies/session):

```
<h1>PHP Cookies and Sessions</h1>
<h2>Cookie Manipulation</h2>
<p>Cookie 'user_preference' is already set to: dark_mode</p>
<p>Retrieved Cookie Value: dark_mode</p>
<hr>
<h2>Session Manipulation</h2>
<p>Session 'views' incremented. Current views: 2</p>
<p>Session 'username' is: GuestUser</p>
<p>Session 'last_activity' updated to: HH:MM:SS</p>
<p><a href='cookies_sessions.php'>Refresh Page to see Session/Cookie
changes</a></p>
```

(If deletion code for cookie/session is uncommented and tested, the output will reflect those actions.)

Laboratory 14: Form Validation, Connecting with MySQL Data using PHP Functions

Title

PHP Form Validation and MySQL Connection

Aim

To develop web forms with client-side and server-side validation and to establish a connection with a MySQL database using PHP functions (PDO or MySQLi) to insert and display data.

Procedure

1. **Database Setup:** Ensure a MySQL database and a table (e.g., `users` with `id`, `name`, `email`, `password`) are set up.
2. **HTML Form Creation:**
    - Create an HTML form (`index.php` or `register.php`) with input fields (text, email, password) and a submit button.
    - Use `method="POST"` for form submission.
3. **PHP Form Processing and Validation:**
    - Check if the form has been submitted (`$_SERVER["REQUEST_METHOD"] == "POST"`).
    - Retrieve form data using `$_POST`.
    - **Validation:**
        - Check for empty fields.
        - Validate email format (`filter_var($email, FILTER_VALIDATE_EMAIL)`).
        - Validate password strength (e.g., minimum length).
        - Sanitize input to prevent SQL injection and XSS (e.g., `htmlspecialchars()`, `trim()`, `stripslashes()`).
    - Store validation errors in an array.
4. **MySQL Database Connection (using PDO - Recommended):**
    - Define database credentials (host, database name, username, password).
    - Use `try-catch` block for database connection to handle errors.
    - Create a new `PDO` object.
    - Set error mode to `PDO::ERRMODE_EXCEPTION`.
5. **Inserting Data:**
    - Prepare an SQL `INSERT` statement using parameterized queries (prepared statements) to prevent SQL injection.
    - Bind values to parameters.
    - Execute the prepared statement.
6. **Displaying Data:**
    - Prepare an SQL `SELECT` statement.
    - Execute the query.
    - Fetch results (e.g., `fetch(PDO::FETCH_ASSOC)` or `fetchAll()`).
    - Loop through the results and display them in an HTML table.

Source Code

```
<?php
// config.php (create this file for database credentials)
/*
```

```
$db_host = 'localhost';
$db_name = 'lab_db';
$db_user = 'root';
$db_pass = 'your_mysql_password'; // IMPORTANT: Change this!
*/

// index.php
require_once 'config.php'; // Include the database configuration

$name = $email = $password = "";
$nameErr = $emailErr = $passwordErr = "";
$successMessage = "";
$users = []; // To store fetched users

// Database connection
try {
    $pdo = new PDO("mysql:host=$db_host;dbname=$db_name", $db_user,
$db_pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // echo "<p style='color:green;'>Database connected successfully!</p>";
// For debugging
} catch (PDOException $e) {
    die("<p style='color:red;'>Database connection failed: " . $e-
>getMessage() . "</p>");
}

// Function to sanitize input
function sanitize_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}

// Process form submission
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Validate Name
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else {
        $name = sanitize_input($_POST["name"]);
        if (!preg_match("/^[a-zA-Z-' ]*$/", $name)) {
            $nameErr = "Only letters and white space allowed";
        }
    }

    // Validate Email
    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else {
        $email = sanitize_input($_POST["email"]);
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $emailErr = "Invalid email format";
        }
    }

    // Validate Password
    if (empty($_POST["password"])) {
        $passwordErr = "Password is required";
    } else {
        $password = sanitize_input($_POST["password"]);
        if (strlen($password) < 6) {
            $passwordErr = "Password must be at least 6 characters long";
        }
        // Hash password for security before storing
        $hashed_password = password_hash($password, PASSWORD_DEFAULT);
```

```
    }

    // If no errors, insert data into database
    if (empty($nameErr) && empty($emailErr) && empty($passwordErr)) {
        try {
            $stmt = $pdo->prepare("INSERT INTO users (name, email, password)
VALUES (:name, :email, :password)");
            $stmt->bindParam(':name', $name);
            $stmt->bindParam(':email', $email);
            $stmt->bindParam(':password', $hashed_password); // Use hashed
password
            $stmt->execute();
            $successMessage = "New user registered successfully!";
            // Clear form fields after successful submission
            $name = $email = $password = "";
        } catch (PDOException $e) {
            // Check for duplicate email error (MySQL error code 23000 for
integrity constraint violation)
            if ($e->getCode() == '23000') {
                $emailErr = "This email is already registered.";
            } else {
                $successMessage = "<span style='color:red;'>Error: " . $e-
>getMessage() . "</span>";
            }
        }
    }
}

// Fetch existing users from the database to display
try {
    $stmt = $pdo->query("SELECT id, name, email FROM users ORDER BY id
DESC");
    $users = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    echo "<p style='color:red;'>Error fetching users: " . $e->getMessage() .
"</p>";
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Registration & Display</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-
color: #f4f4f4; }
        .container { background-color: #fff; padding: 20px; border-radius:
8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 600px; margin: 20px
auto; }
        h1, h2 { color: #333; text-align: center; }
        .error { color: red; font-size: 0.9em; }
        .success { color: green; font-weight: bold; text-align: center; }
        form div { margin-bottom: 15px; }
        label { display: block; margin-bottom: 5px; font-weight: bold; }
        input[type="text"], input[type="email"], input[type="password"] {
            width: calc(100% - 22px); /* Adjust for padding and border */
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        button {
            background-color: #007bff;
            color: white;
            padding: 10px 20px;
```

```
            border: none;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1em;
            width: 100%;
        }
        button:hover { background-color: #0056b3; }
        table { width: 100%; border-collapse: collapse; margin-top: 20px; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Register New User</h1>

        <?php if (!empty($successMessage)): ?>
            <p class="success"><?php echo $successMessage; ?></p>
        <?php endif; ?>

        <form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]); ?>">
            <div>
                <label for="name">Name:</label>
                <input type="text" id="name" name="name" value="<?php echo
htmlspecialchars($name); ?>">
                <span class="error"><?php echo $nameErr; ?></span>
            </div>
            <div>
                <label for="email">Email:</label>
                <input type="email" id="email" name="email" value="<?php echo
htmlspecialchars($email); ?>">
                <span class="error"><?php echo $emailErr; ?></span>
            </div>
            <div>
                <label for="password">Password:</label>
                <input type="password" id="password" name="password"
value="<?php echo htmlspecialchars($password); ?>">
                <span class="error"><?php echo $passwordErr; ?></span>
            </div>
            <div>
                <button type="submit">Register</button>
            </div>
        </form>

        <h2>Registered Users</h2>
        <?php if (!empty($users)): ?>
            <table>
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Name</th>
                        <th>Email</th>
                    </tr>
                </thead>
                <tbody>
                    <?php foreach ($users as $user): ?>
                        <tr>
                            <td><?php echo htmlspecialchars($user['id']);
?></td>
                            <td><?php echo htmlspecialchars($user['name']);
?></td>
                            <td><?php echo htmlspecialchars($user['email']);
?></td>
                        </tr>
                    <?php endforeach; ?>
```

```
                </tbody>
            </table>
        <?php else: ?>
            <p style="text-align: center;">No users registered yet.</p>
        <?php endif; ?>
    </div>
</body>
</html>
```

Database Setup (SQL for `lab_db` and `users` table):

```
-- Connect to your MySQL server as root or a user with privileges
-- mysql -u root -p

-- Create the database if it doesn't exist (from Lab 11)
CREATE DATABASE IF NOT EXISTS lab_db;

-- Use the database
USE lab_db;

-- Create the 'users' table
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE, -- Email must be unique
    password VARCHAR(255) NOT NULL,     -- Store hashed passwords
    reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Input

User inputs through the HTML form fields:

- Name: (e.g., "Jane Doe")
- Email: (e.g., "jane.doe@example.com")
- Password: (e.g., "securepass123")

Expected Output

(Rendered in a web browser)

On initial load or successful registration:

```
<div class="container">
    <h1>Register New User</h1>
    <p class="success">New user registered successfully!</p>
    <form>...</form>
    <h2>Registered Users</h2>
    <table>
        <thead>...</thead>
        <tbody>
            <tr>
                <td>1</td>
                <td>Jane Doe</td>
                <td>jane.doe@example.com</td>
            </tr>
            </tbody>
    </table>
</div>
```

On form submission with validation errors:

```
<div class="container">
    <h1>Register New User</h1>
    <form>
        <div>
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" value="">
            <span class="error">Name is required</span>
        </div>
        <div>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" value="invalid-
email">
            <span class="error">Invalid email format</span>
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password"
value="short">
            <span class="error">Password must be at least 6 characters
long</span>
        </div>
        <div><button type="submit">Register</button></div>
    </form>
    <h2>Registered Users</h2>
    </div>
```

Laboratory 15: Creating PHP Web Applications to Manipulating Data [CRUD Operations] from MySQL Table

Title

PHP Web Application for CRUD Operations

Aim

To develop a comprehensive PHP web application that allows users to perform all four essential data manipulation operations (Create, Read, Update, Delete - CRUD) on records stored in a MySQL database table.

Procedure

1. **Database Setup:** Use the `lab_db` and `users` table from Lab 14, or create a new table (e.g., `products` with `id`, `name`, `description`, `price`).
2. **Application Structure:** Organize the application into multiple PHP files for better modularity:
    - `config.php`: Database connection details.
    - `index.php`: Displays all records (Read operation) and provides links/buttons for Create, Update, Delete.
    - `create.php`: Form for adding new records (Create operation).
    - `edit.php`: Form for editing existing records (Update operation).
    - `delete.php`: Handles deletion of records (Delete operation).
3. **Implement Read (Display All):**
    - On `index.php`, connect to the database.
    - Execute a `SELECT * FROM table_name` query.
    - Display results in an HTML table, with "Edit" and "Delete" links/buttons for each row.
4. **Implement Create (Add New):**
    - On `create.php`, display an HTML form.
    - On form submission, validate input.
    - Use a prepared `INSERT` statement to add data to the database.
    - Redirect back to `index.php` on success.
5. **Implement Update (Edit Existing):**
    - On `edit.php`, retrieve the record ID from the URL (`$_GET['id']`).
    - Fetch the existing record data from the database and pre-fill the HTML form.
    - On form submission, validate input.
    - Use a prepared `UPDATE` statement with a `WHERE` clause based on the record ID.
    - Redirect back to `index.php` on success.
6. **Implement Delete (Remove Record):**
    - On `delete.php`, retrieve the record ID from the URL (`$_GET['id']`).
    - (Optional: Add a confirmation step).
    - Use a prepared `DELETE` statement with a `WHERE` clause based on the record ID.
    - Redirect back to `index.php` on success.
7. **Error Handling and User Feedback:** Use `try-catch` blocks for database operations and display meaningful error/success messages to the user.

Source Code

1. `config.php` (Database connection - same as Lab 14, or adjust for `products` table)

```php
<?php
// config.php
$db_host = 'localhost';
$db_name = 'lab_db'; // Or your specific database name
$db_user = 'root';
$db_pass = 'your_mysql_password'; // IMPORTANT: Change this!

try {
    $pdo = new PDO("mysql:host=$db_host;dbname=$db_name", $db_user,
$db_pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // echo "Database connected successfully!"; // For debugging
} catch (PDOException $e) {
    die("Database connection failed: " . $e->getMessage());
}

// Function to sanitize input
function sanitize_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>
```

2. `index.php` (Read Operation)

```php
<?php
// index.php
require_once 'config.php';

$message = "";
if (isset($_GET['message'])) {
    $message = htmlspecialchars($_GET['message']);
}

// Fetch all products
$products = [];
try {
    $stmt = $pdo->query("SELECT * FROM products ORDER BY id DESC");
    $products = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    $message = "<span style='color:red;'>Error fetching products: " . $e-
>getMessage() . "</span>";
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Product Management (CRUD)</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-
color: #f4f4f4; }
        .container { background-color: #fff; padding: 20px; border-radius:
8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 800px; margin: 20px
auto; }
        h1 { color: #333; text-align: center; }
        .message { text-align: center; margin-bottom: 20px; font-weight:
bold; }
        .success { color: green; }
        .error { color: red; }
        .add-button {
```

```
            display: block;
            width: 150px;
            margin: 0 auto 20px auto;
            padding: 10px 15px;
            background-color: #28a745;
            color: white;
            text-align: center;
            text-decoration: none;
            border-radius: 5px;
            font-weight: bold;
        }
        .add-button:hover { background-color: #218838; }
        table { width: 100%; border-collapse: collapse; margin-top: 20px; }
        th, td { border: 1px solid #ddd; padding: 10px; text-align: left; }
        th { background-color: #f2f2f2; }
        .actions a {
            margin-right: 10px;
            text-decoration: none;
            padding: 5px 10px;
            border-radius: 3px;
        }
        .actions .edit { background-color: #007bff; color: white; }
        .actions .edit:hover { background-color: #0056b3; }
        .actions .delete { background-color: #dc3545; color: white; }
        .actions .delete:hover { background-color: #c82333; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Product Management</h1>

        <?php if (!empty($message)): ?>
            <p class="message <?php echo (strpos($message, 'Error') !==
false) ? 'error' : 'success'; ?>"><?php echo $message; ?></p>
        <?php endif; ?>

        <a href="create.php" class="add-button">Add New Product</a>

        <?php if (!empty($products)): ?>
            <table>
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Name</th>
                        <th>Description</th>
                        <th>Price</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    <?php foreach ($products as $product): ?>
                        <tr>
                            <td><?php echo htmlspecialchars($product['id']);
?></td>
                            <td><?php echo
htmlspecialchars($product['name']); ?></td>
                            <td><?php echo
htmlspecialchars($product['description']); ?></td>
                            <td>$<?php echo
htmlspecialchars(number_format($product['price'], 2)); ?></td>
                            <td class="actions">
                                <a href="edit.php?id=<?php echo
$product['id']; ?>" class="edit">Edit</a>
                                <a href="delete.php?id=<?php echo
$product['id']; ?>" class="delete" onclick="return confirm('Are you sure you
want to delete this product?');">Delete</a>
```

```
                            </td>
                        </tr>
                    <?php endforeach; ?>
                </tbody>
            </table>
        <?php else: ?>
            <p style="text-align: center;">No products found. Add some!</p>
        <?php endif; ?>
    </div>
</body>
</html>
```

3. `create.php` (Create Operation)

```
<?php
// create.php
require_once 'config.php';

$name = $description = $price = "";
$nameErr = $descriptionErr = $priceErr = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Validate Name
    if (empty($_POST["name"])) {
        $nameErr = "Product name is required";
    } else {
        $name = sanitize_input($_POST["name"]);
    }

    // Validate Description (optional)
    $description = sanitize_input($_POST["description"]);

    // Validate Price
    if (empty($_POST["price"])) {
        $priceErr = "Price is required";
    } else {
        $price = sanitize_input($_POST["price"]);
        if (!is_numeric($price) || $price < 0) {
            $priceErr = "Price must be a positive number";
        }
    }

    // If no errors, insert data
    if (empty($nameErr) && empty($descriptionErr) && empty($priceErr)) {
        try {
            $stmt = $pdo->prepare("INSERT INTO products (name, description,
price) VALUES (:name, :description, :price)");
            $stmt->bindParam(':name', $name);
            $stmt->bindParam(':description', $description);
            $stmt->bindParam(':price', $price);
            $stmt->execute();
            header("Location: index.php?message=Product created
successfully!");
            exit();
        } catch (PDOException $e) {
            $nameErr = "Error creating product: " . $e->getMessage(); //
Display error on form
        }
    }
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add New Product</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-
color: #f4f4f4; }
        .container { background-color: #fff; padding: 20px; border-radius:
8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 600px; margin: 20px
auto; }
        h1 { color: #333; text-align: center; }
        .error { color: red; font-size: 0.9em; }
        form div { margin-bottom: 15px; }
        label { display: block; margin-bottom: 5px; font-weight: bold; }
        input[type="text"], input[type="number"], textarea {
            width: calc(100% - 22px);
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        textarea { resize: vertical; min-height: 80px; }
        .buttons { display: flex; justify-content: space-between; gap: 10px;
}
        button, .back-button {
            padding: 10px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1em;
            flex-grow: 1;
            text-align: center;
            text-decoration: none;
        }
        button[type="submit"] { background-color: #28a745; color: white; }
        button[type="submit"]:hover { background-color: #218838; }
        .back-button { background-color: #6c757d; color: white; }
        .back-button:hover { background-color: #5a6268; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Add New Product</h1>

        <form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]); ?>">
            <div>
                <label for="name">Product Name:</label>
                <input type="text" id="name" name="name" value="<?php echo
htmlspecialchars($name); ?>">
                <span class="error"><?php echo $nameErr; ?></span>
            </div>
            <div>
                <label for="description">Description:</label>
                <textarea id="description" name="description"><?php echo
htmlspecialchars($description); ?></textarea>
                <span class="error"><?php echo $descriptionErr; ?></span>
            </div>
            <div>
                <label for="price">Price:</label>
                <input type="number" id="price" name="price" step="0.01"
value="<?php echo htmlspecialchars($price); ?>">
                <span class="error"><?php echo $priceErr; ?></span>
            </div>
            <div class="buttons">
                <button type="submit">Add Product</button>
                <a href="index.php" class="back-button">Back to List</a>
            </div>
```

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

4. `edit.php` (Update Operation)

```
<?php
// edit.php
require_once 'config.php';

$id = $_GET['id'] ?? 0;
$name = $description = $price = "";
$nameErr = $descriptionErr = $priceErr = "";
$productFound = false;

// Fetch existing product data
if ($id > 0) {
    try {
        $stmt = $pdo->prepare("SELECT * FROM products WHERE id = :id");
        $stmt->bindParam(':id', $id, PDO::PARAM_INT);
        $stmt->execute();
        $product = $stmt->fetch(PDO::FETCH_ASSOC);

        if ($product) {
            $productFound = true;
            $name = $product['name'];
            $description = $product['description'];
            $price = $product['price'];
        } else {
            header("Location: index.php?message=Product not found!");
            exit();
        }
    } catch (PDOException $e) {
        header("Location: index.php?message=Error fetching product for edit:
" . urlencode($e->getMessage()));
        exit();
    }
} else {
    header("Location: index.php?message=Invalid product ID!");
    exit();
}

// Process form submission for update
if ($_SERVER["REQUEST_METHOD"] == "POST" && $productFound) {
    // Validate Name
    if (empty($_POST["name"])) {
        $nameErr = "Product name is required";
    } else {
        $name = sanitize_input($_POST["name"]);
    }

    // Validate Description (optional)
    $description = sanitize_input($_POST["description"]);

    // Validate Price
    if (empty($_POST["price"])) {
        $priceErr = "Price is required";
    } else {
        $price = sanitize_input($_POST["price"]);
        if (!is_numeric($price) || $price < 0) {
            $priceErr = "Price must be a positive number";
        }
    }
```

```
    // If no errors, update data
    if (empty($nameErr) && empty($descriptionErr) && empty($priceErr)) {
        try {
            $stmt = $pdo->prepare("UPDATE products SET name = :name,
description = :description, price = :price WHERE id = :id");
            $stmt->bindParam(':name', $name);
            $stmt->bindParam(':description', $description);
            $stmt->bindParam(':price', $price);
            $stmt->bindParam(':id', $id, PDO::PARAM_INT);
            $stmt->execute();
            header("Location: index.php?message=Product updated
successfully!");
            exit();
        } catch (PDOException $e) {
            $nameErr = "Error updating product: " . $e->getMessage(); //
Display error on form
        }
    }
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Edit Product</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-
color: #f4f4f4; }
        .container { background-color: #fff; padding: 20px; border-radius:
8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 600px; margin: 20px
auto; }
        h1 { color: #333; text-align: center; }
        .error { color: red; font-size: 0.9em; }
        form div { margin-bottom: 15px; }
        label { display: block; margin-bottom: 5px; font-weight: bold; }
        input[type="text"], input[type="number"], textarea {
            width: calc(100% - 22px);
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        textarea { resize: vertical; min-height: 80px; }
        .buttons { display: flex; justify-content: space-between; gap: 10px;
}
        button, .back-button {
            padding: 10px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            font-size: 1em;
            flex-grow: 1;
            text-align: center;
            text-decoration: none;
        }
        button[type="submit"] { background-color: #007bff; color: white; }
        button[type="submit"]:hover { background-color: #0056b3; }
        .back-button { background-color: #6c757d; color: white; }
        .back-button:hover { background-color: #5a6268; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Edit Product (ID: <?php echo htmlspecialchars($id); ?>)</h1>
```

```
        <form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]) . "?id=" . htmlspecialchars($id); ?>">
            <div>
                <label for="name">Product Name:</label>
                <input type="text" id="name" name="name" value="<?php echo
htmlspecialchars($name); ?>">
                <span class="error"><?php echo $nameErr; ?></span>
            </div>
            <div>
                <label for="description">Description:</label>
                <textarea id="description" name="description"><?php echo
htmlspecialchars($description); ?></textarea>
                <span class="error"><?php echo $descriptionErr; ?></span>
            </div>
            <div>
                <label for="price">Price:</label>
                <input type="number" id="price" name="price" step="0.01"
value="<?php echo htmlspecialchars($price); ?>">
                <span class="error"><?php echo $priceErr; ?></span>
            </div>
            <div class="buttons">
                <button type="submit">Update Product</button>
                <a href="index.php" class="back-button">Cancel</a>
            </div>
        </form>
    </div>
</body>
</html>
```

5. `delete.php` (Delete Operation)

```
<?php
// delete.php
require_once 'config.php';

$id = $_GET['id'] ?? 0;

if ($id > 0) {
    try {
        $stmt = $pdo->prepare("DELETE FROM products WHERE id = :id");
        $stmt->bindParam(':id', $id, PDO::PARAM_INT);
        $stmt->execute();
        if ($stmt->rowCount() > 0) {
            header("Location: index.php?message=Product deleted
successfully!");
        } else {
            header("Location: index.php?message=Product not found or already
deleted!");
        }
        exit();
    } catch (PDOException $e) {
        header("Location: index.php?message=Error deleting product: " .
urlencode($e->getMessage()));
        exit();
    }
} else {
    header("Location: index.php?message=Invalid product ID for deletion!");
    exit();
}
?>
```

Database Setup (SQL for `products` table - if not using `users` table):

```
-- Connect to your MySQL server as root or a user with privileges
-- mysql -u root -p

-- Create the database if it doesn't exist
CREATE DATABASE IF NOT EXISTS lab_db;

-- Use the database
USE lab_db;

-- Create the 'products' table
CREATE TABLE IF NOT EXISTS products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Input

- **Create:** Product Name, Description, Price via `create.php` form.
- **Read:** N/A (Viewing `index.php`).
- **Update:** Product ID (via URL parameter), then updated Product Name, Description, Price via `edit.php` form.
- **Delete:** Product ID (via URL parameter, usually from `index.php` link).

Expected Output

(Rendered in a web browser)

`index.php` (after adding a few products):

```
<div class="container">
    <h1>Product Management</h1>
    <p class="message success">Product created successfully!</p> <a
href="create.php" class="add-button">Add New Product</a>
    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Description</th>
                <th>Price</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>1</td>
                <td>Laptop X</td>
                <td>Powerful laptop for gaming.</td>
                <td>$1200.00</td>
                <td class="actions">
                    <a href="edit.php?id=1" class="edit">Edit</a>
                    <a href="delete.php?id=1" class="delete" onclick="return
confirm('Are you sure you want to delete this product?');">Delete</a>
                </td>
            </tr>
            <tr>
                <td>2</td>
                <td>Mouse Pro</td>
```

```
                <td>Ergonomic wireless mouse.</td>
                <td>$45.50</td>
                <td class="actions">
                    <a href="edit.php?id=2" class="edit">Edit</a>
                    <a href="delete.php?id=2" class="delete" onclick="return
confirm('Are you sure you want to delete this product?');">Delete</a>
                </td>
            </tr>
        </tbody>
    </table>
</div>
```

`create.php` (form to add a new product):

```
<div class="container">
    <h1>Add New Product</h1>
    <form>
        <div><label>Product Name:</label><input type="text"
name="name"></div>
        <div><label>Description:</label><textarea
name="description"></textarea></div>
        <div><label>Price:</label><input type="number" name="price"></div>
        <div class="buttons">
            <button type="submit">Add Product</button>
            <a href="index.php" class="back-button">Back to List</a>
        </div>
    </form>
</div>
```

`edit.php` (form pre-filled with existing data for ID=1):

```
<div class="container">
    <h1>Edit Product (ID: 1)</h1>
    <form>
        <div><label>Product Name:</label><input type="text" name="name"
value="Laptop X"></div>
        <div><label>Description:</label><textarea name="description">Powerful
laptop for gaming.</textarea></div>
        <div><label>Price:</label><input type="number" name="price"
value="1200.00"></div>
        <div class="buttons">
            <button type="submit">Update Product</button>
            <a href="index.php" class="back-button">Cancel</a>
        </div>
    </form>
</div>
```

After successful deletion (redirects to `index.php` with message): (The deleted product will no longer appear in the table on `index.php`.)

```
<div class="container">
    <h1>Product Management</h1>
    <p class="message success">Product deleted successfully!</p>
    </div>
```