JavaScript Array Methods — Quick Notes

Method	Definition	Syntax	Example	Output
`filter()`	Returns a new array with elements that pass a test	`array.filter(callback)`	`[1,2,3,4].filter(x => x%2==0)`	`[2,4]`
`map()`	Creates a new array by modifying each element	`array.map(callback)`	`[1,2,3].map(x => x*2)`	`[2,4,6]`
`slice()`	Returns part of an array (no change in original)	`array.slice(start, end)`	`[10,20,30].slice(1,2)`	`[20]`
`splice()`	Changes array by adding/removing items	`array.splice(start, deleteCount, item...)`	`arr = [1,2,3]; arr.splice(1,1,10)`	`[1,10,3]`
`reduce()`	Reduces array to a single value	`array.reduce(callback, init)`	`[1,2,3].reduce((a,b) => a+b)`	`6`
`some()`	Returns `true` if any element passes test	`array.some(callback)`	`[1,3,5].some(x => x%2==0)`	`false`
`find()`	Returns first element that passes test	`array.find(callback)`	`[4,5,6].find(x => x > 4)`	`5`
`indexOf()`	Returns index of element, or -1	`array.indexOf(value)`	`[10,20].indexOf(20)`	`1`

Method	Definition	Syntax	Example	Output
`includes()`	Checks if value exists in array	`array.includes(value)`	`[1,2,3].includes(2)`	`true`
`concat()`	Merges two arrays	`array1.concat(array2)`	`[1,2].concat([3,4])`	`[1,2,3,4]`
`sort()`	Sorts elements (as strings by default)	`array.sort()`	`[3,1,2].sort()`	`[1,2,3]` (use `sort((a,b)⇒a-b)` for numbers)

PHP - DETAILED NOTES

Overview of Web Development

Definition:
Web development is the process of building websites or web applications that run on the internet. It includes designing web pages, writing code, and connecting to databases.

Types of Web Development:

- **Frontend** (Client-side): What users see (HTML, CSS, JavaScript)
- **Backend** (Server-side): The logic behind the scenes (PHP, MySQL, etc.)

Server vs. Client-Side

Aspect	Client-Side	Server-Side
Runs on	User's browser	Web server
Languages	HTML, CSS, JavaScript	PHP, Python, Node.js, etc.
Access to DB	Cannot access database	Can access database
Speed	Fast (no server communication)	Slower (due to server interaction)

Example:

- **Client-side**: A button that changes color using JavaScript.
- **Server-side**: A PHP script that checks a login username and password.

HTML Basics

Definition:
HTML (HyperText Markup Language) is the standard language used to create and structure web pages.

Basic HTML Example:

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

```
<head>
  <title>My First Page</title>
</head>
<body>
  <h1>Hello World</h1>
  <p>This is my first web page.</p>
</body>
</html>
```

Introduction to PHP and MySQL

PHP Definition:
PHP (Hypertext Preprocessor) is a server-side scripting language used to create dynamic web pages.

MySQL Definition:
MySQL is a database system used to store, retrieve, and manage data in web applications.

Simple PHP Example:

```php
<?php
echo "Welcome to PHP!";
?>
```

Simple MySQL + PHP Example:

```php
<?php
// Connect to MySQL
$conn = mysqli_connect("localhost", "root", "", "mydatabase");

// Check connection
if ($conn) {
    echo "Connected successfully";
} else {
    echo "Connection failed";
}
?>
```

Installation of XAMPP/WAMP

Definition:
XAMPP is a software package that bundle together:

- Apache (Web server)
- MySQL (Database server)
- PHP (Scripting language)
- phpMyAdmin (Database management tool)

These tools allow you to run a PHP website **locally** (on your own computer).

Installation Steps (XAMPP):

1. Download XAMPP from https://www.apachefriends.org
2. Install it using default settings. (Location: `C:\xampp`)
3. Open **XAMPP Control Panel** (From Windows Search Box `WINDOWS KEY + S` to open Search)
4. Click **Start** next to Apache and MySQL

Introduction to Apache, MySQL, and PHP

Component	Role in Web Development
Apache	Web server software that handles HTTP requests (e.g., loading web pages)
MySQL	Database server that stores your data (e.g., user info, products)
PHP	Server-side scripting language used to generate dynamic content

Example Workflow:

1. **Apache** listens for a browser request (e.g., `localhost/index.php`)
2. **PHP** code runs on the server to handle the request
3. **MySQL** is used if the script needs to fetch/store data
4. Apache sends the result back to the browser

Configuring a Development Environment

Definition:
Setting up your system to write and run PHP code locally using XAMPP or WAMP.

Steps (using XAMPP):

1. Place your project files in the folder:
   `C:\xampp\htdocs\your_project_folder`
2. Access your project in the browser like this:
   `http://localhost/your_project_folder`
3. Create a simple test file to verify setup:
   `C:\xampp\htdocs\test.php`

```php
<?php
echo "PHP is working!";
?>
```

4. Visit `http://localhost/test.php` in your browser
   You should see:
   `PHP is working!`
   `Create your folder inside of htdocs`

PHP Documentation

If you want to read the PHP Documentation go to www.php.net/docs.php
Select your language and Start reading (its written in plain English so it easy to understand)

PHP Syntax

Definition:
PHP syntax refers to the basic rules and structure used to write PHP code. PHP scripts are executed on the server and can be embedded in HTML.

Basic PHP Rules:

- PHP code starts with `<?php` and ends with `?>` or `<? ...code... ?>`
- Each statement ends with a semicolon `;`

- Comments can be written using `//`, `#`, or `/* */`

Example 1: Simple PHP Script

```php
<?php
echo "Hello, World!";
?>
```

Output:

```
Hello, World!
```

Example 2: PHP inside HTML

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

<h1>Welcome</h1>
<p>
<?php
echo "This is written using PHP!";
?>
</p>

</body>
</html>
```

Output in browser:

```
Welcome
This is written using PHP!
```

Variables in PHP

Definition:
A **variable** in PHP is used to store data like text, numbers, or arrays that can be used

and changed throughout the script.

Rules:

- All variables start with a `$` symbol.
- Variable names are **case-sensitive**.
- A variable name must start with a **letter or underscore** (`_`), not a number.

Example 1: Declaring and Using Variables

```php
<?php
$name = "Alice";
$age = 21;

echo "Name: " . $name . "<br>";
echo "Age: " . $age;
?>
```

Output:

```
Name: Alice
Age: 21
```

Example 2: Changing Variable Values

```php
<?php
$number = 5;
echo $number;

$number = 10;
echo "<br>" . $number;
?>
```

Data Types in PHP

Definition:
A **data type** tells PHP what kind of data is stored in a variable (like number, text,

true/false, etc.). PHP is **loosely typed**, meaning you don't need to declare the type; PHP detects it automatically.

Common PHP Data Types:

Type	Description	Example
String	Text data	`"Hello"`
Integer	Whole numbers	`123`
Float	Decimal numbers	`3.14`
Boolean	True or false	`true`, `false`
Array	Collection of values	`[1, 2, 3]`
Object	Instance of a class (OOP)	`$car = new Car();`
NULL	Variable with no value	`$x = null;`

Example: Using Different Data Types

```php
<?php
$name = "John";         // String
$age = 25;              // Integer
$price = 19.99;         // Float
$isStudent = true;      // Boolean
$colors = array("Red", "Blue", "Green"); // Array
$nothing = null;        // NULL

echo $name . "<br>";
echo $age . "<br>";
echo $price . "<br>";
echo $isStudent . "<br>"; // Output: 1 (true is shown as 1)
echo $colors[0] . "<br>"; // Output: Red
?>
```

Operators in PHP

Definition:
Operators are symbols used to perform operations on variables and values like calculations, comparisons, or logical tests.

Types of PHP Operators:

1. **Arithmetic Operators** – for calculations
2. **Assignment Operators** – for assigning values
3. **Comparison Operators** – for comparing values
4. **Logical Operators** – for combining conditions

---

1. Arithmetic Operators

Operator	Description	Example (`$a = 10, $b = 5`)	Result
`+`	Addition	`$a + $b`	15
`-`	Subtraction	`$a - $b`	5
`*`	Multiplication	`$a * $b`	50
`/`	Division	`$a / $b`	2
`%`	Modulus (remainder)	`$a % $b`	0

---

2. Assignment Operators

Operator	Example	Meaning
`=`	`$x = 5`	Assign 5 to $x
`+=`	`$x += 3`	`$x = $x + 3`
`-=`	`$x -= 2`	`$x = $x - 2`

---

3. Comparison Operators

Operator	Description	Example (`$a = 10, $b = 5`)	Result
`==`	Equal to	`$a == $b`	false
`!=`	Not equal	`$a != $b`	true
`>`	Greater than	`$a > $b`	true
`<`	Less than	`$a < $b`	false

Operator	Description	Example (`$a = 10, $b = 5`)	Result
`⩾`	Greater than or equal	`$a ⩾ $b`	true
`⩽`	Less than or equal	`$a ⩽ $b`	false

---

4. Logical Operators

Operator	Description	Example (`$x = true, $y = false`)	Result
`&&`	And	`$x && $y`	false
`||`	Or	`$x || $y`	true
`!`	Not	`!$x`	false

Simple Example Using Operators

```php
<?php
$a = 10;
$b = 5;

// Arithmetic
echo "Add: " . ($a + $b) . "<br>";
echo "Divide: " . ($a / $b) . "<br>";

// Comparison
echo "Is equal? " . ($a == $b) . "<br>";

// Logical
$x = true;
$y = false;
echo "x AND y: " . ($x && $y) . "<br>";
?>
```

Output:

```
Add: 15
Divide: 2
Is equal?
x AND y:
```

(`false` shows as blank)

Control Structures in PHP

Definition:
Control structures are used to **control the flow** of the program, like making decisions (`if`) or repeating actions (`while`, `for`).

---

1. `if`, `else if`, `else` – Conditional Statements

Syntax:

```
if (condition) {
  // code if condition is true
} elseif (another_condition) {
  // code if this one is true
} else {
  // code if none are true
}
```

Example:

```
<?php
$marks = 85;

if ($marks ⩾ 90) {
    echo "Grade: A";
} elseif ($marks ⩾ 75) {
    echo "Grade: B";
} else {
    echo "Grade: C";
}
?>
```

---

2. `switch` – For Multiple Choices

Syntax:

```
switch (value) {
  case option1:
```

```
    // code
    break;
  case option2:
    // code
    break;
  default:
    // code
}
```

Example:

```php
<?php
$day = "Monday";

switch ($day) {
    case "Monday":
        echo "Start of the week!";
        break;
    case "Friday":
        echo "Almost weekend!";
        break;
    default:
        echo "Just another day.";
}
?>
```

---

3. `while` Loop – Repeats while condition is true

Example:

```php
<?php
$i = 1;
while ($i ⩽ 3) {
    echo "Number: $i <br>";
    $i++;
}
?>
```

---

4. `for` Loop – Repeats a fixed number of times

Example:

```php
<?php
for ($i = 1; $i <= 3; $i++) {
    echo "Count: $i <br>";
}
?>
```

---

5. `foreach` – For looping through arrays

Example:

```php
<?php
$fruits = array("Apple", "Banana", "Cherry");

foreach ($fruits as $fruit) {
    echo $fruit . "<br>";
}
?>
```

Defining and Using Functions in PHP

Definition:
A **function** is a block of code that performs a specific task and can be reused. It runs only when called.

Why use functions?

- To **organize code**
- To **avoid repetition**
- To **perform tasks with input and return output**

Syntax:

```php
function functionName() {
    // code to run
}

functionName(); // calling the function
```

Example 1: Function without parameters

```php
<?php
function greet() {
    echo "Hello from PHP!";
}

greet(); // Calling the function
?>
```

Output:

```
Hello from PHP!
```

Example 2: Function with parameters

```php
<?php
function greetUser($name) {
    echo "Hello, " . $name;
}

greetUser("Alice"); // Output: Hello, Alice
?>
```

Example 3: Function with return value

```php
<?php
function add($a, $b) {
    return $a + $b;
}

$result = add(5, 3);
echo "Sum: " . $result; // Output: Sum: 8
?>
```

Function Scope in PHP

Definition:
Scope refers to where a variable can be accessed or used in your PHP code.

In PHP, there are **3 types of variable scope**:

---

1. Local Scope

A variable declared **inside a function** can only be used **inside that function**.

Example:

```php
<?php
function showMessage() {
    $msg = "Hello!";
    echo $msg;
}

showMessage(); // Output: Hello!
// echo $msg; // ❌ Error: $msg is undefined outside
?>
```

---

2. Global Scope

A variable declared **outside** any function is called a **global variable**. To use it **inside** a function, use the `global` keyword.

Example:

```php
<?php
$name = "John";

function greet() {
    global $name;
    echo "Hi, $name!";
}

greet(); // Output: Hi, John!
?>
```

---

3. Static Scope

When a variable inside a function is declared as `static`, its value is **remembered** between function calls.

Example:

```php
<?php
function counter() {
    static $count = 0;
    $count++;
    echo $count . "<br>";
}

counter(); // Output: 1
counter(); // Output: 2
counter(); // Output: 3
?>
```

Including Files in PHP

Definition:
Including files in PHP allows you to **reuse code** (like headers, footers, menus, or database connections) in multiple pages without rewriting it.

PHP Include Statements:

Statement	Description
`include`	Includes the file. If the file is not found, it shows a warning but continues.
`require`	Includes the file. If the file is not found, it shows a fatal error and stops.
`include_once`	Includes the file only once (even if called multiple times).
`require_once`	Same as `require`, but prevents multiple inclusions.

Example 1: Using `include`

File: `header.php`

```php
<?php
echo "<h1>Welcome to My Website</h1><hr>";
?>
```

File: `index.php`

```php
<?php
include 'header.php';
echo "<hr><br>" . "This is the homepage.";
?>
```

Output:

```
Welcome to My Website
---------------------
This is the homepage.
```

Example 2: Using `require`

```php
<?php
require 'header.php';  // Required file
echo "Main content here.";
?>
```

If `header.php` is missing, PHP will stop the script with an error.

Example 3: Using `include_once`

```php
<?php
include_once 'header.php';
include_once 'header.php'; // This won't include again
?>
```

Table: Common PHP Built-in Functions

Function	Description	Example Usage	Output (Example)
`strlen()`	Returns the length of a string	`strlen("Hello")`	`5`
`strtoupper()`	Converts a string to uppercase	`strtoupper("php")`	`PHP`
`strtolower()`	Converts a string to lowercase	`strtolower("PHP")`	`php`
`str_replace()`	Replaces text in a string	`str_replace("world", "PHP", "Hello world")`	`Hello PHP`
`strpos()`	Finds position of first occurrence of a substring	`strpos("Hello", "e")`	`1`
`round()`	Rounds a float number	`round(4.6)`	`5`
`ceil()`	Rounds up a float number	`ceil(4.2)`	`5`
`floor()`	Rounds down a float number	`floor(4.9)`	`4`
`array_sum()`	Returns the sum of array values	`array_sum([1, 2, 3])`	`6`
`count()`	Counts number of elements in an array	`count(["a", "b", "c"])`	`3`
`in_array()`	Checks if a value exists in an array	`in_array("a", ["a", "b"])`	`true` (1)
`isset()`	Checks if a variable is set	`isset($x)`	`true` or `false`
`empty()`	Checks if a variable is empty	`empty("")`	`true`
`date()`	Formats a date/time	`date("Y-m-d")`	`2025-07-03` (example)
`time()`	Returns current Unix timestamp	`time()`	`Unix time number`
`explode()`	Splits a string into an array	`explode(",", "a,b,c")`	`["a", "b", "c"]`
`implode()`	Joins array elements into a string	`implode("-", ["a", "b", "c"])`	`a-b-c`
`var_dump()`	Dumps variable info (type + value)	`var_dump(5.5)`	`float(5.5)`

Function	Description	Example Usage	Output (Example)
`print_r()`	Prints human-readable info about a variable	`print_r(["a", "b"])`	`Array ( [0] ⇒ a ...)`

> **Note**
>
> These are the most common ones there are more than 1000+, Check PHP Documentation for those functions.

Working with Arrays in PHP

Definition:
An **array** is a variable that can store **multiple values** at once. Each value in the array is called an **element**.

Types of Arrays in PHP

Type	Description	Example
Indexed Array	Stores values with numeric keys	`["Apple", "Banana", "Cherry"]`
Associative	Stores values with named keys	`["name" ⇒ "Alice", "age" ⇒ 20]`
Multidimensional	Array containing other arrays	`[["A", "B"], ["C", "D"]]`

---

1. Indexed Array

Example:

```php
<?php
$fruits = array("Apple", "Banana", "Cherry");
```

```php
echo $fruits[1]; // Output: Banana
?>
```

---

2. Associative Array (Dictionary from Python)

Example:

```php
<?php
$person = array("name" => "Alice", "age" => 20);
echo "Name: " . $person["name"]; // Output: Name: Alice
?>
```

---

3. Multidimensional Array

Example:

```php
<?php
$marks = array(
  array(85, 90),
  array(78, 88)
);
echo $marks[1][0]; // Output: 78
?>
```

Useful Array Operations

Task	Example Code	Output
Add element (push)	`array_push($fruits, "Orange");`	Adds to end
Count elements	`count($fruits);`	Number of items
Loop through array	`foreach ($fruits as $fruit) { ... }`	Prints all items
Merge arrays	`array_merge($a1, $a2);`	Combines arrays
Remove last element	`array_pop($fruits);`	Removes last item
Remove first element	`array_shift($fruits);`	Removes first item
Sort values	`sort($fruits);`	Alphabetical order

Example: Loop Through Array

```php
<?php
$colors = array("Red", "Green", "Blue");

foreach ($colors as $color) {
    echo $color . "<br>";
}
?>
```

Output:

```
Red
Green
Blue
```

Here is the full explanation for:

Array Functions in PHP

Definition:
Array functions are built-in PHP functions used to manipulate, search, and handle arrays easily.

Table of Common Array Functions

Function	Description	Example Code	Output Example
`count()`	Returns the number of elements	`count([1, 2, 3])`	`3`
`array_push()`	Adds item(s) to the end	`array_push($a, "dog")`	Adds "dog" to array
`array_pop()`	Removes the last item	`array_pop($a)`	Removes last element

Function	Description	Example Code	Output Example
`array_shift()`	Removes the first item	`array_shift($a)`	Removes first element
`array_unshift()`	Adds item(s) to the beginning	`array_unshift($a, "cat")`	Adds to front
`in_array()`	Checks if a value exists	`in_array("apple", $a)`	`true` / `false`
`array_merge()`	Merges two or more arrays	`array_merge($a1, $a2)`	Combined array
`array_unique()`	Removes duplicate values	`array_unique([1,2,2,3])`	`[1,2,3]`
`array_reverse()`	Reverses the array	`array_reverse([1,2,3])`	`[3,2,1]`
`array_keys()`	Returns all the keys	`array_keys(["a" ⇒ 1, "b" ⇒ 2])`	`["a", "b"]`
`array_values()`	Returns all the values	`array_values(["a" ⇒ 1, "b" ⇒ 2])`	`[1, 2]`
`sort()`	Sorts indexed array (ascending)	`sort($a)`	Sorts values
`rsort()`	Sorts indexed array (descending)	`rsort($a)`	Reverse sort
`asort()`	Sorts associative array by value	`asort($a)`	Keeps key-value match
`ksort()`	Sorts associative array by key	`ksort($a)`	Sorts by keys
`array_slice()`	Extracts a portion of an array	`array_slice($a, 1, 2)`	Gets part of array
`array_splice()`	Removes/replaces array elements	`array_splice($a, 1, 2)`	Removes part
`array_search()`	Searches for a value and returns its key	`array_search("apple", $a)`	Key/index of "apple"

Example 1: `array_push()` and `count()`

```php
<?php
$animals = ["Cat", "Dog"];
```

```php
array_push($animals, "Tiger");
echo count($animals); // Output: 3
?>
```

Example 2: `in_array()` and `array_merge()`

```php
<?php
$colors1 = ["Red", "Green"];
$colors2 = ["Blue", "Yellow"];
$all = array_merge($colors1, $colors2);

if (in_array("Blue", $all)) {
    echo "Blue is in the list!";
}
?>
```

Output:

```
Blue is in the list!
```

Array Functions – Examples

---

1. `count()`

```php
<?php
$nums = [10, 20, 30];
echo count($nums); // Output: 3
?>
```

---

2. `array_push()`

```php
<?php
$fruits = ["Apple", "Banana"];
array_push($fruits, "Mango");
print_r($fruits);
```

```php
// Output: Array ( [0] => Apple [1] => Banana [2] => Mango )
?>
```

---

3. array_pop()

```php
<?php
$fruits = ["Apple", "Banana", "Mango"];
array_pop($fruits);
print_r($fruits);
// Output: Array ( [0] => Apple [1] => Banana )
?>
```

---

4. array_shift()

```php
<?php
$fruits = ["Apple", "Banana", "Mango"];
array_shift($fruits);
print_r($fruits);
// Output: Array ( [0] => Banana [1] => Mango )
?>
```

---

5. array_unshift()

```php
<?php
$fruits = ["Banana", "Mango"];
array_unshift($fruits, "Apple");
print_r($fruits);
// Output: Array ( [0] => Apple [1] => Banana [2] => Mango )
?>
```

---

6. in_array()

```php
<?php
$fruits = ["Apple", "Banana"];
if (in_array("Banana", $fruits)) {
    echo "Found!";
```

```php
}
// Output: Found!
?>
```

---

7. array_merge()

```php
<?php
$a = [1, 2];
$b = [3, 4];
$c = array_merge($a, $b);
print_r($c);
// Output: Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
?>
```

---

8. array_unique()

```php
<?php
$nums = [1, 2, 2, 3];
$unique = array_unique($nums);
print_r($unique);
// Output: Array ( [0] => 1 [1] => 2 [3] => 3 )
?>
```

---

9. array_reverse()

```php
<?php
$nums = [1, 2, 3];
$reversed = array_reverse($nums);
print_r($reversed);
// Output: Array ( [0] => 3 [1] => 2 [2] => 1 )
?>
```

---

10. array_keys()

```php
<?php
$person = ["name" => "John", "age" => 30];
```

```php
$keys = array_keys($person);
print_r($keys);
// Output: Array ( [0] => name [1] => age )
?>
```

---

11. `array_values()`

```php
<?php
$person = ["name" => "John", "age" => 30];
$values = array_values($person);
print_r($values);
// Output: Array ( [0] => John [1] => 30 )
?>
```

---

12. `sort()`

```php
<?php
$nums = [3, 1, 2];
sort($nums);
print_r($nums);
// Output: Array ( [0] => 1 [1] => 2 [2] => 3 )
?>
```

---

13. `rsort()`

```php
<?php
$nums = [1, 3, 2];
rsort($nums);
print_r($nums);
// Output: Array ( [0] => 3 [1] => 2 [2] => 1 )
?>
```

---

14. `asort()` – Sort by value and keep key

```php
<?php
$person = ["a" => 30, "b" => 20];
```

```php
asort($person);
print_r($person);
// Output: Array ( [b] => 20 [a] => 30 )
?>
```

---

15. `ksort()` – Sort by key

```php
<?php
$person = ["b" => "Banana", "a" => "Apple"];
ksort($person);
print_r($person);
// Output: Array ( [a] => Apple [b] => Banana )
?>
```

---

16. `array_slice()`

```php
<?php
$fruits = ["Apple", "Banana", "Mango", "Orange"];
$sliced = array_slice($fruits, 1, 2);
print_r($sliced);
// Output: Array ( [0] => Banana [1] => Mango )
?>
```

---

17. `array_splice()`

```php
<?php
$fruits = ["Apple", "Banana", "Mango"];
array_splice($fruits, 1, 1);
print_r($fruits);
// Output: Array ( [0] => Apple [1] => Mango )
?>
```

---

18. `array_search()`

```php
<?php
$fruits = ["Apple", "Banana", "Mango"];
```

```php
$key = array_search("Banana", $fruits);
echo $key;
// Output: 1
?>
```

String Manipulation

Definition:
String manipulation means changing or handling string values using code — like cutting, joining, replacing, or formatting.

---

1. Concatenation (Joining Strings)

Definition: Use `.` to join strings in PHP.

Example:

```php
<?php
$first = "Hello";
$second = "World";
echo $first . " " . $second; // Output: Hello World
?>
```

---

2. Extracting a Substring – `substr()`

Definition: Returns a portion of a string.

Example:

```php
<?php
echo substr("Welcome", 0, 4); // Output: Welc
?>
```

---

3. Replace Part of String – `str_replace()`

Definition: Replaces all occurrences of a word or character.

Example:

```php
<?php
echo str_replace("world", "PHP", "Hello world"); // Output: Hello PHP
?>
```

---

4. Split String into Array – `explode()`

Definition: Splits a string by a separator.

Example:

```php
<?php
$data = "apple,banana,mango";
$fruits = explode(",", $data);
print_r($fruits); // Output: Array ( [0] ⇒ apple [1] ⇒ banana [2] ⇒
mango )
?>
```

---

5. Join Array into String – `implode()`

Definition: Joins array elements into a single string.

Example:

```php
<?php
$colors = ["red", "green", "blue"];
echo implode("-", $colors); // Output: red-green-blue
?>
```

String Functions

---

1. `strlen()`

Definition: Returns number of characters in a string.

Example:

```php
<?php
echo strlen("Hello"); // Output: 5
?>
```

---

2. `strtoupper()` / `strtolower()`

Definition: Converts string to upper/lowercase.

Example:

```php
<?php
echo strtoupper("php"); // Output: PHP
?>
```

---

3. `ucfirst()` / `ucwords()`

Definition: Capitalize first letter of string / each word.

Example:

```php
<?php
echo ucwords("hello php"); // Output: Hello Php
?>
```

---

4. `trim()`

Definition: Removes spaces from both ends of a string.

Example:

```php
<?php
echo trim("  hello  "); // Output: hello
?>
```

---

5. `strrev()`

Definition: Reverses the characters in a string.

Example:

```php
<?php
echo strrev("hello"); // Output: olleh
?>
```

Handling Form Data in PHP

Definition:
Handling form data means receiving input from users (like text, email, etc.) sent through an HTML `<form>` and processing it with PHP.

When a form is submitted, data is sent to a PHP file using either the **GET** or **POST** method. PHP uses `$_GET` and `$_POST` ***superglobals*** to access the submitted values.

Example: Simple Form Handling

Step 1: HTML Form (`form.html`)

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

<form action="submit.php" method="post">
  Name: <input type="text" name="username"><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

Step 2: PHP File to Handle Form (`submit.php`)

```php
<?php
$name = $_POST['username'];
echo "Hello, " . $name;
?>
```

Explanation:

- `action="submit.php"` → sends data to `submit.php`
- `method="post"` → sends data securely using POST
- `$_POST['username']` → accesses the input value from the form

Output (if user typed "Alice"):

```
Hello, Alice
```

> **SUPERGLOBAL Variable Note**
>
> No need to run the code at the moment in the below its just for understanding what is Super Global Variable

---

Super Global Variables in PHP

Definition:
Super global variables are **built-in variables** in PHP that are **always accessible**, anywhere in the script (inside or outside functions) and used to handle user input, server data, sessions, cookies, etc.

List of Common Super Global Variables

Super Global	Description
`$_GET`	Contains form data sent using `GET` method
`$_POST`	Contains form data sent using `POST` method
`$_REQUEST`	Contains both `$_GET`, `$_POST`, and `$_COOKIE`
`$_SERVER`	Contains server and request information
`$_SESSION`	Stores session variables
`$_COOKIE`	Stores cookies
`$_FILES`	Handles uploaded files

Super Global	Description
`$_ENV`	Environment variables (from server config)
`$_GLOBALS`	Accesses global variables from anywhere

Example: Using `$_POST`, `$_GET`, `$_SERVER`, `$_REQUEST`

HTML Form (form.html)

```
<form action="process.php" method="get">
  Name: <input type="text" name="username">
  <input type="submit" value="Send">
</form>
```

PHP File (`process.php`)

```
<?php
// Using $_GET
echo "Hello, " . $_GET['username'] . "<br>";

// Using $_SERVER
echo "You are using: " . $_SERVER['HTTP_USER_AGENT'] . "<br>";

// Using $_REQUEST
echo "Received via REQUEST: " . $_REQUEST['username'];
?>
```

Output (`if user enters "Alice"`):

```
Hello, Alice
You are using: Mozilla/5.0...
Received via REQUEST: Alice
```

GET vs. POST Methods in PHP

Definition:
Both **GET** and **POST** are methods used to **send form data** to a PHP script. They are used in the `<form>` tag with the `method` attribute.

Difference Between GET and POST

Feature	GET	POST
Data Visible	Yes, in URL	No, sent in request body
Max Data Size	Limited (~2000 characters)	No size limit (depends on server)
Use Case	Search, filters (safe data)	Login, password (sensitive data)
Syntax	`$_GET['name']`	`$_POST['name']`
Bookmarkable	Yes	No

Example: GET Method

HTML Form (`form_get.html`)

```
<form action="get_handler.php" method="get">
  Name: <input type="text" name="username">
  <input type="submit">
</form>
```

PHP (`get_handler.php`)

```
<?php
echo "You entered: " . $_GET['username'];
?>
```

URL after submit:
`http://localhost/get_handler.php?username=Alice`
Output:

```
You entered: Alice
```

Example: POST Method

HTML Form (`form_post.html`)

```
<form action="post_handler.php" method="post">
  Name: <input type="text" name="username">
  <input type="submit">
</form>
```

PHP (`post_handler.php`)

```
<?php
echo "You entered: " . $_POST['username'];
?>
```

URL stays clean:
`http://localhost/post_handler.php`
Output (after typing Alice):

```
You entered: Alice
```

Validating and Sanitizing User Input

Definition:

- **Validation** checks if the input is correct (e.g., is it an email? is it empty? is it a number?).
- **Sanitization** cleans the input (removes or escapes unwanted characters) to prevent **security risks** like XSS or SQL injection.

---

1. Validation

Example: Check if a name is empty

```
<?php
$name = $_POST['username'];

if (empty($name)) {
    echo "Name is required";
} else {
    echo "Hello, " . $name;
}
?>
```

Example: Validate an email

```php
<?php
$email = $_POST['email'];

if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Valid email: " . $email;
} else {
    echo "Invalid email format";
}
?>
```

---

2. Sanitization

Example: Sanitize a string (remove tags and extra characters)

```php
<?php
$raw = "<script>alert('Hacked');</script> Hello!";
$clean = filter_var($raw, FILTER_SANITIZE_STRING);
echo $clean;
// Output: alert('Hacked'); Hello!
?>
```

Example: Sanitize email

```php
<?php
$email = "test@@example.com";
$cleanEmail = filter_var($email, FILTER_SANITIZE_EMAIL);
echo $cleanEmail; // Output: test@example.com
?>
```

Best Practice: Combine Validation and Sanitization

```php
<?php
$email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);

if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
```

```
    echo "Email is clean and valid.";
} else {
    echo "Invalid email.";
}
?>
```

Connecting PHP to MySQL

Definition:
To work with databases, PHP uses built-in functions (like `mysqli_connect`) to connect to a **MySQL server**. This connection lets you run SQL queries and manage data.

Syntax (Using MySQLi)

```
mysqli_connect(server, username, password, database);
```

- `server` = usually `"localhost"` (means your own computer)
- `username` = MySQL username (default: `"root"` in XAMPP)
- `password` = MySQL password (default: `""` empty in XAMPP)
- `database` = name of your MySQL database

Example: Connect to MySQL Database

```
<?php
$host = "localhost";
$user = "root";
$pass = "";
$db = "testdb";

// Connect to MySQL
$conn = mysqli_connect($host, $user, $pass, $db);

// Check connection
if ($conn) {
    echo "Connected successfully";
} else {
    echo "Connection failed: " . mysqli_connect_error();
```

```
}
?>
```

Notes:

- You must create the database (`testdb`) in ***phpMyAdmin*** before running this script.
- Use `mysqli_connect_error()` to see the connection error if it fails.

Executing Queries in PHP (with MySQL)

Definition:
Once connected to MySQL, you can run SQL queries (like SELECT, INSERT, UPDATE, DELETE) using PHP's `mysqli_query()` function.

Syntax:

```
mysqli_query($connection, $sql);
```

- `$connection` = your database connection object
- `$sql` = the SQL query as a string

Example: Create and Insert Data

```
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");

// Insert a new user
$sql = "INSERT INTO users (name, email) VALUES ('Alice',
'alice@example.com')";
$result = mysqli_query($conn, $sql);

if ($result) {
    echo "Data inserted successfully";
} else {
    echo "Error: " . mysqli_error($conn);
```

```
}
?>
```

Example: SELECT Query

```php
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");

$sql = "SELECT * FROM users";
$result = mysqli_query($conn, $sql);

// Check if data exists
if (mysqli_num_rows($result) > 0) {
    echo "Query executed successfully";
} else {
    echo "No data found";
}
?>
```

Retrieving Data in PHP (from MySQL)

Definition:
To retrieve (read) data from a MySQL database, use a `SELECT` query with `mysqli_query()`, and then fetch the results using `mysqli_fetch_assoc()` or similar functions.

Step-by-Step Example

```php
<?php
// Connect to the database
$conn = mysqli_connect("localhost", "root", "", "testdb");

// SQL SELECT query
$sql = "SELECT name, email FROM users";
$result = mysqli_query($conn, $sql);

// Check and fetch rows
if (mysqli_num_rows($result) > 0) {
    while ($row = mysqli_fetch_assoc($result)) {
        echo "Name: " . $row["name"] . "<br>";
```

```
        echo "Email: " . $row["email"] . "<hr>";
    }
} else {
    echo "No users found.";
}
?>
```

Functions Used:

- `mysqli_query()` → runs the SELECT query
- `mysqli_num_rows()` → checks if data exists
- `mysqli_fetch_assoc()` → fetches one row at a time as an associative array (`$row["column_name"]`)

Displaying Data in Web Pages

Definition:
After retrieving data from a MySQL database using PHP, you can display it in an HTML page using `echo`, loops, and HTML tags like `<table>`, `<div>`, or `<p>`.

Example: Display MySQL Data in an HTML Table

```
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");

$sql = "SELECT name, email FROM users";
$result = mysqli_query($conn, $sql);
?>

<!DOCTYPE html>
<html>
<head>
    <title>Users List</title>
</head>
<body>
    <h2>User Data</h2>
    <table border="1">
        <tr>
            <th>Name</th>
            <th>Email</th>
```

```
        </tr>

        <?php
        if (mysqli_num_rows($result) > 0) {
            while ($row = mysqli_fetch_assoc($result)) {
                echo "<tr>";
                echo "<td>" . $row["name"] . "</td>";
                echo "<td>" . $row["email"] . "</td>";
                echo "</tr>";
            }
        } else {
            echo "<tr><td colspan='2'>No data found</td></tr>";
        }
        ?>

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

How it works:

- PHP connects to the database
- Executes a `SELECT` query
- Embeds PHP inside HTML to loop and print data inside the table rows

---

CRUD Operations in PHP and MySQL

Definition:
CRUD stands for the 4 basic operations used in web apps to manage data:

1. **Create** – Add new records
2. **Read** – View records
3. **Update** – Change existing records
4. **Delete** – Remove records

We use PHP to handle these actions and MySQL to store the data.

Setup (Assume this table in MySQL)

```sql
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100)
);
```

---

1. Create (Insert Data)

```php
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");
$sql = "INSERT INTO users (name, email) VALUES ('Alice',
'alice@example.com')";
mysqli_query($conn, $sql);
echo "Data inserted";
?>
```

---

2. Read (View Data)

```php
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");
$result = mysqli_query($conn, "SELECT * FROM users");

while ($row = mysqli_fetch_assoc($result)) {
    echo "Name: " . $row['name'] . " - Email: " . $row['email'] . "<br>";
}
?>
```

---

3. Update (Edit Data)

```php
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");
$sql = "UPDATE users SET email='alice123@example.com' WHERE name='Alice'";
mysqli_query($conn, $sql);
echo "Data updated";
?>
```

---

4. Delete (Remove Data)

```php
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");
$sql = "DELETE FROM users WHERE name='Alice'";
mysqli_query($conn, $sql);
echo "Data deleted";
?>
```

Note: For real applications, use forms with `POST` method and always validate user input.

Session Management in PHP

Definition:
A **session** is a way to store user information (like login data) across multiple pages. Unlike cookies, session data is stored on the server.

To use sessions in PHP, you must start the session using `session_start()` before sending any HTML output.

---

1. Starting a Session

Syntax:

```php
session_start();
```

This must be at the top of every page that uses session data.

---

2. Storing Data in Session

Example:

```php
<?php
session_start();
$_SESSION['username'] = "Alice";
```

```php
echo "Session started and value stored.";
?>
```

---

3. Accessing Session Data on Another Page

Example:

```php
<?php
session_start();
echo "Welcome, " . $_SESSION['username'];
?>
```

Output:

```
Welcome, Alice
```

---

4. Removing a Session Variable

Example:

```php
<?php
session_start();
unset($_SESSION['username']); // removes only 'username'
?>
```

---

5. Destroying the Entire Session

Example:

```php
<?php
session_start();
session_destroy(); // clears all session data
?>
```

Understanding Sessions and Cookies

1. Sessions

Definition:
A **session** stores user data **on the server** for the duration of a user's visit or login.

- Session data is lost when the browser is closed or `session_destroy()` is called.
- Uses a **session ID** to identify users.

Use Case: Login info, user preferences, cart data

Example: Store and show session

```php
<?php
session_start();
$_SESSION['user'] = "Alice";
echo "Session set: " . $_SESSION['user'];
?>
```

---

2. Cookies

Definition:
A **cookie** stores data **in the user's browser** for a set time.

- It can stay for days or months even after the browser is closed.
- Created using `setcookie()`.

Use Case: Remember username, tracking, preferences

Syntax:

```
setcookie(name, value, expire_time);
```

Example: Set and read a cookie

```php
<?php
setcookie("username", "Alice", time() + 3600); // lasts 1 hour
echo $_COOKIE['username']; // Output: Alice
?>
```

Key Differences

Feature	Session	Cookie
Stored in	Server	Client (browser)
Size limit	Large	Small (~4KB)
Duration	Until browser closes	Until expiry time
Use case	Secure login/session	Preferences, remember user

Creating and Managing Sessions in PHP

Definition:
To **create** and **manage sessions** in PHP, you use `session_start()` to begin, and the `$_SESSION` superglobal to store and access session data across pages.

---

1. Start a Session

Must be called at the top of the page before any output.

```php
<?php
session_start();
?>
```

---

2. Store Data in a Session

```php
<?php
session_start();
$_SESSION['username'] = "Alice";
$_SESSION['logged_in'] = true;
echo "Session data stored.";
?>
```

---

3. Access Session Data on Another Page

```php
<?php
session_start();
```

```php
if (isset($_SESSION['username'])) {
    echo "Welcome, " . $_SESSION['username'];
} else {
    echo "User not logged in.";
}
?>
```

---

4. Remove a Single Session Variable

```php
<?php
session_start();
unset($_SESSION['username']); // Removes only 'username'
?>
```

---

5. Destroy All Session Data

```php
<?php
session_start();
session_destroy(); // Deletes all session variables
?>
```

Notes:

- Use sessions for **secure data** like login state, cart contents, etc.
- Always start with `session_start()` on every page that reads or writes session data.

User Authentication in PHP

Definition:
User authentication is the process of verifying a user's identity — typically using a **username and password**. If the credentials match, the user is allowed access.

Authentication is often implemented using **PHP + MySQL + sessions**.

Step-by-Step Example: Simple Login System

1. Database Table (users)

Create a table in MySQL:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(100),
  password VARCHAR(100)
);
```

Insert a sample user (password is plain text for demo):

```
INSERT INTO users (username, password) VALUES ('admin', '12345');
```

---

2. HTML Login Form (`login.html`)

```
<form action="login.php" method="post">
  Username: <input type="text" name="username"><br>
  Password: <input type="password" name="password"><br>
  <input type="submit" value="Login">
</form>
```

---

3. PHP Login Handler (`login.php`)

```
<?php
session_start();

$conn = mysqli_connect("localhost", "root", "", "testdb");

$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM users WHERE username='$username' AND
password='$password'";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) == 1) {
    $_SESSION['username'] = $username;
    echo "Login successful. Welcome, " . $username;
```

```
} else {
    echo "Invalid username or password.";
}
?>
```

---

4. Logout Script (`logout.php`)

```
<?php
session_start();
session_destroy();
echo "Logged out successfully.";
?>
```

---

5. Protect Page (`dashboard.php`)

```
<?php
session_start();

if (!isset($_SESSION['username'])) {
    echo "Access denied. Please log in.";
    exit;
}

echo "Welcome to the dashboard, " . $_SESSION['username'];
?>
```

Building a User Login System in PHP

Definition:
A **user login system** lets users log in securely with a username and password. PHP verifies credentials from a **MySQL database** and uses **sessions** to keep users logged in.

Step-by-Step Login System

---

1. Create Database and Table

```sql
CREATE DATABASE login_db;

USE login_db;

CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(100),
  password VARCHAR(100) -- Plain text for now (hash later)
);
```

Insert a test user:

```sql
INSERT INTO users (username, password) VALUES ('admin', '12345');
```

---

2. HTML Login Form (`login.html`)

```html
<form action="login.php" method="post">
  <label>Username:</label>
  <input type="text" name="username"><br>

  <label>Password:</label>
  <input type="password" name="password"><br>

  <input type="submit" value="Login">
</form>
```

---

3. PHP Login Handler (`login.php`)

```php
<?php
session_start();

$conn = mysqli_connect("localhost", "root", "", "login_db");

$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM users WHERE username='$username' AND
password='$password'";
$result = mysqli_query($conn, $sql);
```

```php
if (mysqli_num_rows($result) == 1) {
    $_SESSION['username'] = $username;
    header("Location: dashboard.php"); // Redirect to dashboard
} else {
    echo "Invalid login credentials.";
}
?>
```

---

4. Dashboard Page (`dashboard.php`)

```php
<?php
session_start();

if (!isset($_SESSION['username'])) {
    echo "Access denied. Please log in.";
    exit;
}

echo "Welcome, " . $_SESSION['username'];
?>
<br><a href="logout.php">Logout</a>
```

---

5. Logout Page (`logout.php`)

```php
<?php
session_start();
session_destroy();
echo "You have been logged out.";
?>
```

Summary

- `login.html` → User enters credentials
- `login.php` → Validates from DB, sets session
- `dashboard.php` → Accessible only after login
- `logout.php` → Ends session

Password Hashing in PHP

Definition:
Password hashing means converting a password into a secure format using a one-way function, so the original password is not stored in the database. PHP provides `password_hash()` and `password_verify()` for this purpose.

This is important for **security** — never store plain text passwords.

Step-by-Step Example

---

1. Hashing a Password Before Storing

```php
<?php
$password = "mypassword";
$hashed = password_hash($password, PASSWORD_DEFAULT);
echo $hashed; // Output: something like $2y$10$...
?>
```

- `PASSWORD_DEFAULT` uses the latest strong algorithm (like bcrypt)
- Each time, the hash is different even for the same password

---

2. Store the `$hashed` value in the database (not the plain password)

```sql
INSERT INTO users (username, password) VALUES ('admin', '$2y$10$...');
```

---

3. Verifying Password During Login

```php
<?php
$conn = mysqli_connect("localhost", "root", "", "login_db");

$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM users WHERE username='$username'";
```

```php
$result = mysqli_query($conn, $sql);

if ($row = mysqli_fetch_assoc($result)) {
    // Check hashed password
    if (password_verify($password, $row['password'])) {
        echo "Login successful";
    } else {
        echo "Wrong password";
    }
} else {
    echo "User not found";
}
?>
```

Summary:

- `password_hash()` → Hash the password (for signup)
- `password_verify()` → Check plain password against the hash (for login)

Error Handling in PHP

Definition:
Error handling in PHP means detecting and responding to errors (like file not found, database connection failed, etc.) in a controlled way — instead of showing raw errors to users.

PHP provides:

- **Error types**: notices, warnings, fatal errors
- **Functions**: `error_reporting()`, `try...catch`, `die()`, `trigger_error()`

Basic Error Handling Methods

---

1. Using `die()` for Critical Errors

```php
<?php
$conn = mysqli_connect("localhost", "root", "", "wrong_db");
```

```php
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
?>
```

Output:

```
Connection failed: Unknown database 'wrong_db'
```

---

2. Custom Error Reporting

```php
<?php
error_reporting(E_ALL); // Show all errors
ini_set("display_errors", 1);
echo $x; // Undefined variable (notice)
?>
```

---

3. `try...catch` with `Exception`

```php
<?php
function divide($a, $b) {
    if ($b == 0) {
        throw new Exception("Cannot divide by zero");
    }
    return $a / $b;
}

try {
    echo divide(10, 0);
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
}
?>
```

Output:

```
Error: Cannot divide by zero
```

---

4. `trigger_error()` for Custom Errors

```php
<?php
$age = -5;

if ($age < 0) {
    trigger_error("Age cannot be negative");
}
?>
```

File Handling in PHP

Definition:
File handling means reading from and writing to files on the server using PHP functions like `fopen()`, `fread()`, `fwrite()`, and `fclose()`.

PHP can read text files, write data, create new files, and more.

Reading Files

---

1. `fopen()` + `fread()` – Read file contents

```php
<?php
$file = fopen("demo.txt", "r"); // open for reading
$content = fread($file, filesize("demo.txt")); // read whole file
fclose($file); // close the file
echo $content;
?>
```

> Assumes `demo.txt` already exists and contains text like: `Hello World`.

---

2. `file_get_contents()` – Quick way to read file

```php
<?php
$text = file_get_contents("demo.txt");
```

```
echo $text;
?>
```

Writing Files

---

1. `fopen()` + `fwrite()` – Write to a file

```php
<?php
$file = fopen("demo.txt", "w"); // open for writing (overwrite)
fwrite($file, "This is new content");
fclose($file);
echo "Data written to file.";
?>
```

> If `demo.txt` does not exist, it will be created.

---

2. `file_put_contents()` – Quick way to write to a file

```php
<?php
file_put_contents("demo.txt", "Quick write content");
echo "Done writing.";
?>
```

Modes for `fopen()`:

Mode	Description
`r`	Read only
`w`	Write only (overwrite)
`a`	Write only (append)
`r+`	Read and write
`a+`	Read and write (append)

Uploading Files in PHP

Definition:
File uploading allows users to upload files (images, PDFs, etc.) from a form to your server. PHP handles this using the `$_FILES` superglobal.

Step-by-Step File Upload Example

1. Create an HTML Form

```
<form action="upload.php" method="post" enctype="multipart/form-data">
  Select file: <input type="file" name="myfile"><br>
  <input type="submit" value="Upload">
</form>
```

- `enctype="multipart/form-data"` is required for file uploads.

---

2. Handle Upload in PHP (`upload.php`)

```
<?php
if ($_FILES['myfile']['error'] == 0) {
    $name = $_FILES['myfile']['name'];
    $tmp = $_FILES['myfile']['tmp_name'];
    $target = "uploads/" . $name;

    if (move_uploaded_file($tmp, $target)) {
        echo "File uploaded successfully.";
    } else {
        echo "File upload failed.";
    }
} else {
    echo "No file selected or error occurred.";
}
?>
```

---

3. Folder Setup

- Create a folder named `uploads` in the same directory as `upload.php`.
- Make sure the folder has **write permission**.

Basic File Info

Use `$_FILES['myfile']` to access:

Key	Description
`name`	Original filename
`tmp_name`	Temporary file path on server
`size`	File size in bytes
`type`	MIME type (e.g. `image/png`)
`error`	Upload error code (0 = no error)

Security in PHP

Definition:
Security in PHP means writing code that protects your application from attacks like SQL injection, XSS, file upload abuse, and unauthorized access. PHP has built-in methods to help secure input, output, sessions, and database queries.

Common Security Measures

---

1. Input Validation and Sanitization

Why: Prevents users from sending harmful code.

Example: Sanitize a string

```php
<?php
$name = filter_var($_POST['name'], FILTER_SANITIZE_STRING);
?>
```

---

2. SQL Injection Prevention

Why: Prevents attackers from modifying your SQL queries.

Bad (vulnerable):

```
$sql = "SELECT * FROM users WHERE username = '$user'";
```

Good (secure using prepared statements):

```
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");

$stmt = mysqli_prepare($conn, "SELECT * FROM users WHERE username = ?");
mysqli_stmt_bind_param($stmt, "s", $user);
mysqli_stmt_execute($stmt);
?>
```

---

3. Cross-Site Scripting (XSS) Protection

Why: Prevents scripts from being injected into your HTML.

Use `htmlspecialchars()` when displaying user input:

```
<?php
echo htmlspecialchars($user_input);
?>
```

---

4. Password Hashing

Why: Never store plain text passwords.

Use:

```
$passwordHash = password_hash("mypassword", PASSWORD_DEFAULT);
```

Verify:

```
if (password_verify($input, $passwordHash)) { /* success */ }
```

---

5. Secure File Uploads

Check file type before saving:

```
$type = $_FILES['file']['type'];
if ($type == "image/jpeg" || $type == "image/png") {
    move_uploaded_file($_FILES['file']['tmp_name'], "uploads/" .
$_FILES['file']['name']);
}
```

---

6. Session Security

- Always use `session_start()` at the top.
- Regenerate session IDs after login:

```
session_regenerate_id();
```

- Destroy session on logout:

```
session_destroy();
```

SQL Injection Prevention in PHP

Definition:
SQL injection is a security attack where an attacker sends harmful SQL code through user input fields to break or control your database.

Prevention means writing your queries safely — by **not directly inserting user input** into SQL. Instead, use **prepared statements**.

Vulnerable Code (Not Safe)

```
<?php
$username = $_POST['username'];
$password = $_POST['password'];

$conn = mysqli_connect("localhost", "root", "", "testdb");
$sql = "SELECT * FROM users WHERE username = '$username' AND password =
```

```
'$password'";
$result = mysqli_query($conn, $sql);
?>
```

If someone enters `' OR 1=1 --` as input, it can bypass login.

Secure Code Using Prepared Statements

```
<?php
$conn = mysqli_connect("localhost", "root", "", "testdb");

$username = $_POST['username'];
$password = $_POST['password'];

$stmt = mysqli_prepare($conn, "SELECT * FROM users WHERE username = ? AND
password = ?");
mysqli_stmt_bind_param($stmt, "ss", $username, $password);
mysqli_stmt_execute($stmt);

$result = mysqli_stmt_get_result($stmt);

if (mysqli_num_rows($result) == 1) {
    echo "Login successful";
} else {
    echo "Invalid credentials";
}
?>
```

Why This is Safe:

- The `?` placeholders are automatically escaped
- `mysqli_stmt_bind_param()` binds values securely
- No SQL code from the user can be executed

XSS Protection in PHP

Definition:
XSS (Cross-Site Scripting) is when attackers inject JavaScript into your web pages (e.g., `<script>alert('hacked')</script>`). PHP can prevent this by escaping special HTML characters.

Use `htmlspecialchars()` to Prevent XSS

Example:

```php
<?php
$user_input = "<script>alert('Hacked');</script>";
$safe_output = htmlspecialchars($user_input);
echo $safe_output;
?>
```

Output:

```
&lt;script&gt;alert('Hacked');&lt;/script&gt;
```

Now, the script will **not execute**, and will show as plain text instead.

Securing File Uploads in PHP

Definition:
When allowing users to upload files, you must check the file type, size, and restrict paths to prevent malicious uploads (like `.php` files).

Secure File Upload Example

```php
<?php
if (isset($_FILES['myfile']) && $_FILES['myfile']['error'] == 0) {
    $fileName = $_FILES['myfile']['name'];
    $fileTmp  = $_FILES['myfile']['tmp_name'];
    $fileSize = $_FILES['myfile']['size'];
    $fileType = mime_content_type($fileTmp);

    // Allow only images
    $allowedTypes = ['image/jpeg', 'image/png'];

    if (in_array($fileType, $allowedTypes) && $fileSize <= 2 * 1024 *
1024) {
        move_uploaded_file($fileTmp, "uploads/" . basename($fileName));
        echo "File uploaded safely.";
    } else {
        echo "Invalid file type or size.";
```

```
    }
} else {
    echo "Upload failed.";
}
?>
```

Key Security Measures:

- Use `mime_content_type()` to check actual file type
- Set a size limit (e.g., 2MB)
- Store files in a safe `uploads/` folder
- Rename files before saving to prevent overwriting

Advanced PHP Concepts

This section introduces more structured and powerful PHP features often used in larger applications.

Object-Oriented PHP (OOP)

Definition:
Object-Oriented Programming (OOP) in PHP is a way of organizing code using **classes** and **objects**, instead of just functions and variables. It helps make code reusable, structured, and easier to manage.

Key Terms:

Term	Description
Class	A blueprint for objects
Object	An instance of a class
Property	A variable inside a class
Method	A function inside a class
$this	Refers to the current object

Example: Basic Class and Object

```php
<?php
class Person {
    public $name; // Property

    // Method
    public function sayHello() {
        return "Hello, " . $this->name;
    }
}

// Create an object
$user = new Person();
$user->name = "Alice";
echo $user->sayHello(); // Output: Hello, Alice
?>
```

How It Works:

- `class Person` defines the structure
- `$user = new Person();` creates an object
- `$user->name = "Alice";` sets a property
- `$user->sayHello();` calls a method

Classes in PHP

Definition:
A **class** is a blueprint for creating objects. It defines **properties** (variables) and **methods** (functions) that the object will have.

Syntax:

```php
class ClassName {
    // properties
    // methods
}
```

Example: Define a Simple Class

```php
<?php
class Car {
    public $brand = "Toyota";

    public function showBrand() {
        echo "Brand: " . $this->brand;
    }
}
?>
```

Explanation:

- `class Car` defines a new class named `Car`
- `$brand` is a property
- `showBrand()` is a method
- `$this->brand` refers to the property of the current object

You can use this class by creating an object (which we will do in the next topic).

Objects in PHP

Definition:
An **object** is an instance of a class. It is created using the `new` keyword and gives access to the class's **properties** and **methods**.

Syntax:

```php
$objectName = new ClassName();
```

Example: Create and Use an Object

```php
<?php
class Car {
    public $brand = "Toyota";

    public function showBrand() {
        echo "Brand: " . $this->brand;
    }
}

// Create an object from the class
$myCar = new Car();

// Access method and property using the object
$myCar->showBrand(); // Output: Brand: Toyota
?>
```

Explanation:

- `new Car()` creates an object of the class
- `$myCar->brand` accesses a property
- `$myCar->showBrand()` calls a method

Inheritance in PHP

Definition:
Inheritance allows one class (child) to use the properties and methods of another class (parent). It helps reuse and extend code.

A child class uses the `extends` keyword to inherit from a parent class.

Example: Inheritance in Action

```php
<?php
// Parent class
class Animal {
    public function makeSound() {
        echo "Some sound";
    }
}
```

```php
// Child class inherits from Animal
class Dog extends Animal {
    public function makeSound() {
        echo "Bark";
    }
}

$myDog = new Dog();
$myDog->makeSound(); // Output: Bark
?>
```

Explanation:

- `Dog` extends `Animal` and inherits its method
- The `makeSound()` method is **overridden** in `Dog`
- `$myDog->makeSound()` uses the method from the child class

PHP's Built-in OOP Features

Definition:
PHP supports **Object-Oriented Programming (OOP)** with built-in features that help in creating structured, reusable, and scalable applications.

Main OOP Features in PHP

Feature	Description
Classes & Objects	Core of OOP. Objects are created from classes.
Properties	Variables defined inside a class.
Methods	Functions defined inside a class.
Inheritance	One class can inherit from another using `extends`.
Access Modifiers	Controls visibility: `public`, `protected`, `private`.
Constructors	Special method that runs automatically when an object is created.
Destructors	Method that runs when an object is destroyed.
Static Members	Properties or methods that belong to the class, not the object.

Feature	Description
Constants	Use `const` inside a class to define unchangeable values.
Interfaces	A way to define common methods for multiple classes.
Abstract Classes	Cannot be instantiated directly, only extended.
Traits	Used to reuse methods in multiple unrelated classes.

Simple Example Using Constructor, Property, Method

```php
<?php
class User {
    public $name;

    // Constructor (runs automatically)
    public function __construct($username) {
        $this->name = $username;
    }

    public function greet() {
        return "Hello, " . $this->name;
    }
}

$user1 = new User("Alice");
echo $user1->greet(); // Output: Hello, Alice
?>
```

Introduction to PHP Frameworks

Definition:
A **PHP framework** is a set of **pre-built tools, libraries, and structure** that helps developers build web applications **faster and more securely**. It follows best practices like MVC (Model–View–Controller) and reduces the need to write repetitive code.

Why Use a PHP Framework?

Benefit	Description
Code organization	Keeps logic, display, and data separated (MVC)
Faster development	Provides built-in tools for routing, forms, DB access
Security features	Protects against SQL injection, CSRF, XSS
Reusable components	Use libraries for sessions, validation, mail, etc.
Scalability	Easier to maintain and expand large applications

Common PHP Frameworks

Framework	Key Features	IN SYLLABUS
Laravel	Most popular, elegant syntax, MVC, artisan CLI	YES
CodeIgniter	Lightweight, simple, fast setup	YES
Symfony	Stable and powerful for enterprise use	NO
CakePHP	Rapid development, convention-based	NO
Yii	Fast, secure, component-based	NO
Slim	Minimalist framework for APIs and micro apps	NO

Example Code: Routing in Laravel (very basic)

```
// web.php (Laravel route file)
Route::get('/hello', function () {
    return 'Hello from Laravel!';
});
```

This is much simpler than writing raw PHP routing logic manually.

Overview of Popular PHP Frameworks

Definition:
A **PHP framework** is a structured platform for building web applications. It provides reusable code, tools, and best practices like MVC (Model–View–Controller) to make development faster and more secure.

1. Laravel

- **Most popular** PHP framework
- Uses **MVC architecture**
- Elegant syntax and powerful features
- Comes with:
    - **Blade** template engine (for views)
    - **Eloquent ORM** (for working with databases easily)
    - **Artisan CLI** (command line tools)
    - **Built-in authentication and routing**

Use Case: Large-scale, modern web apps

---

2. CodeIgniter

- Lightweight and **easy to learn**
- Also uses **MVC**
- Very fast and good for small to medium apps
- Simple setup, minimal configuration
- Works well on shared hosting

Use Case: Quick projects or apps with fewer dependencies

Basic MVC Architecture Concepts

Definition:
MVC stands for **Model–View–Controller**. It is a design pattern used in PHP frameworks (like Laravel, CodeIgniter) to **organize code** into 3 separate parts:

Components of MVC

Part	Role
Model	Handles data and database logic
View	Displays data (HTML output shown to the user)
Controller	Receives user input, processes it, and calls Model or View

Real-World Analogy:

- **Model** = The database (where the information lives)
- **View** = The screen (what the user sees)
- **Controller** = The logic (the traffic controller that connects them)

Simple Flow:

1. User visits a URL → handled by the **Controller**
2. Controller asks the **Model** for data
3. Model fetches from database and returns it
4. Controller passes data to the **View**
5. View shows the data to the user

Example (Concept Only – Pseudo PHP)

Controller (`UserController.php`)

```
$user = UserModel::getUser(1);      // get data from model
loadView("profile", $user);         // pass data to view
```

Model (`UserModel.php`)

```
class UserModel {
    public static function getUser($id) {
        // query database and return user
    }
}
```

View (`profile.php`)

```
<h1>Welcome, <?php echo $user['name']; ?></h1>
```

Why MVC Is Useful

- Keeps code **organized** and **modular**

- Makes it easier to **maintain and test**
- Separates **logic**, **data**, and **design**

What is Laravel?

Laravel is a **free**, **open-source PHP web framework** used to build **modern web applications**. It follows the **MVC (Model-View-Controller)** architecture and simplifies tasks like routing, authentication, sessions, and caching.

Laravel is known for its **elegant syntax**, **developer-friendly structure**, and built-in tools like **Artisan CLI**, **Eloquent ORM**, **Blade templating**, and **routing system**.

Why use Laravel?

- Cleaner and faster development
- Secure and scalable
- Built-in authentication and session handling
- Database operations using Eloquent ORM
- Useful CLI (Artisan) for common tasks

Installation (Basic Setup)

> You must have **Composer** installed before installing Laravel.

```
composer create-project laravel/laravel myApp
```

This command creates a Laravel project named `myApp`.

Simple Laravel Example

Let's create a basic **route** that returns "Hello, Laravel":

File: `routes/web.php`

```php
<?php

use Illuminate\Support\Facades\Route;

Route::get('/hello', function () {
    return 'Hello, Laravel';
});
```

Output:

Visiting `http://localhost:8000/hello` in the browser will display:

```
Hello, Laravel
```

Run Laravel Project

To run the Laravel server, go to your project directory and use:

```
php artisan serve
```

Default server URL: `http://localhost:8000`

What is Routing in Laravel?

Routing in Laravel is how you define the **URLs** (web addresses) your application will respond to. Each route is linked to a function or controller that tells Laravel what to do when that URL is visited.

Laravel routes are defined in the `routes/web.php` file for web pages and `routes/api.php` for APIs.

Types of Routes

1. **Basic Route** – Responds to a URL with a function.
2. **Route with Parameters** – Pass data in the URL.
3. **Named Routes** – Give a route a custom name.
4. **Routes to Controllers** – Route calls a controller method.

1. Basic Route Example

```
Route::get('/welcome', function () {
    return 'Welcome to Laravel Routing!';
});
```

Visit: `http://localhost:8000/welcome`
Output: `Welcome to Laravel Routing!`

---

2. Route with Parameter

```
Route::get('/user/{name}', function ($name) {
    return "Hello, $name";
});
```

Visit: `http://localhost:8000/user/Amit`
Output: `Hello, Amit`

---

3. Named Route

```
Route::get('/about', function () {
    return 'About Us';
})->name('about.page');
```

You can now use this name (`about.page`) to generate URLs:

```
$url = route('about.page');
```

---

4. Route to Controller

First, create a controller:

```
php artisan make:controller PageController
```

Then, in `PageController.php`:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PageController extends Controller
{
    public function home() {
        return "This is the Home Page.";
    }
}
```

Define route in `web.php`:

```php
Route::get('/home', [PageController::class, 'home']);
```

Visit: `http://localhost:8000/home`
Output: `This is the Home Page.`

Creating and Using Controllers

What is a Controller?

A **controller** is a PHP class where you organize the logic of your web application. It handles user requests and returns responses.

How to Create a Controller

Use Artisan command:

```
php artisan make:controller MyController
```

This creates `app/Http/Controllers/MyController.php`.

Example: Basic Controller

File: `app/Http/Controllers/MyController.php`

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class MyController extends Controller
{
    public function showWelcome()
    {
        return 'Welcome from Controller!';
    }
}
```

Route: `routes/web.php`

```php
use App\Http\Controllers\MyController;

Route::get('/welcome', [MyController::class, 'showWelcome']);
```

Visit: `http://localhost:8000/welcome`
Output: `Welcome from Controller!`

Passing Data to Views

What is a View?

A **view** is an HTML file with embedded Blade syntax (Laravel's templating engine). It displays the content sent from the controller.

Step-by-Step Example

1. Create a View

File: `resources/views/greeting.blade.php`

```php
<!DOCTYPE html>
<html>
<head>
    <title>Greeting</title>
</head>
<body>
```

```
    <h1>Hello, {{ $name }}!</h1>
</body>
</html>
```

2. Controller Method

```php
public function greetUser()
{
    $name = "Amit";
    return view('greeting', ['name' => $name]);
}
```

3. Route

```php
Route::get('/greet', [MyController::class, 'greetUser']);
```

Visit: `http://localhost:8000/greet`
Output:
`Hello, Amit!`

Route Parameters

What are Route Parameters?

Route parameters allow you to **pass values from the URL** into your route or controller method.

Example with Controller

Route:

```php
Route::get('/user/{name}', [MyController::class, 'showUser']);
```

Controller Method:

```php
public function showUser($name)
{
    return "User name is: $name";
}
```

Visit: `http://localhost:8000/user/John`
Output: `User name is: John`

Introduction to Blade Templating

Blade is Laravel's built-in **templating engine**. It allows embedding PHP code in HTML files with a cleaner and easier syntax. Blade files use the `.blade.php` extension and are stored in the `resources/views` directory.

Blade File Example

File: `resources/views/hello.blade.php`

```php
<!DOCTYPE html>
<html>
<head>
    <title>Hello Page</title>
</head>
<body>
    <h1>Hello from Blade</h1>
</body>
</html>
```

Route:

```php
Route::get('/hello', function () {
    return view('hello');
});
```

Output: Shows a page with "Hello from Blade"

Using Blade Directives

Blade directives are special commands that start with `@`.

Common Directives

- `@if`, `@elseif`, `@else`, `@endif`
- `@foreach`, `@endforeach`
- `@for`, `@endfor`
- `@include`, `@extends`, `@section`, `@yield`
- `{{ }}` to display variables

Example:

```
<h1>Welcome, {{ $name }}</h1>

@if($age ⩾ 18)
    <p>You are an adult.</p>
@else
    <p>You are a minor.</p>
@endif
```

Template Inheritance

Blade lets you define a **master layout** and reuse it in other pages using `@extends`, `@section`, and `@yield`.

Step 1: Create Layout

File: `resources/views/layouts/main.blade.php`

```
<!DOCTYPE html>
<html>
<head>
    <title>My Site</title>
</head>
<body>
    <header>Site Header</header>

    <div class="content">
        @yield('content')
    </div>

    <footer>Site Footer</footer>
```

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

Step 2: Extend Layout

File: `resources/views/home.blade.php`

```
@extends('layouts.main')

@section('content')
    <h2>Home Page</h2>
    <p>This is the home page content.</p>
@endsection
```

Route:

```
Route::get('/home', function () {
    return view('home');
});
```

Displaying Data in Views

To pass data from the controller or route to a view, use the `view()` function.

Controller Method

```
public function showProfile()
{
    $name = "Amit";
    $email = "amit@example.com";
    return view('profile', ['name' => $name, 'email' => $email]);
}
```

Blade View (`profile.blade.php`)

```
<h1>Profile</h1>
<p>Name: {{ $name }}</p>
<p>Email: {{ $email }}</p>
```

Route:

```
Route::get('/profile', [MyController::class, 'showProfile']);
```

Output:

```
Name: Amit
Email: amit@example.com
```

Overview of Eloquent ORM

Eloquent ORM is Laravel's built-in **Object Relational Mapper**. It allows you to interact with your database using **PHP classes** instead of writing raw SQL. Each **model** represents a table in your database.

Defining Models

Creating a Model

Use Artisan command:

```
php artisan make:model Student
```

This creates: `app/Models/Student.php`

Example Model

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    protected $fillable = ['name', 'email'];
}
```

Here, `fillable` protects against mass assignment.

Basic CRUD Operations

1. Create

```
Student::create([
    'name' => 'Amit',
    'email' => 'amit@example.com'
]);
```

Or using object:

```
$student = new Student;
$student->name = 'Amit';
$student->email = 'amit@example.com';
$student->save();
```

---

2. Read

```
$all = Student::all(); // Get all students

$one = Student::find(1); // Find by ID

$search = Student::where('name', 'Amit')->first(); // Filtered query
```

---

3. Update

```
$student = Student::find(1);
$student->email = 'newemail@example.com';
$student->save();
```

---

4. Delete

```
$student = Student::find(1);
$student->delete();
```

Database Migrations

Migrations are version-controlled files to create and modify tables.

Create a Migration

```
php artisan make:migration create_students_table
```

Then in the file:

```
Schema::create('students', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->timestamps();
});
```

Run it:

```
php artisan migrate
```

Relationships

Eloquent supports relationships between tables (models).

1. One to One

Example: A User has one Profile

```
// In User.php
public function profile()
{
    return $this->hasOne(Profile::class);
}

// In Profile.php
public function user()
{
    return $this->belongsTo(User::class);
}
```

2. One to Many

Example: A Post has many Comments

```php
// In Post.php
public function comments()
{
    return $this->hasMany(Comment::class);
}

// In Comment.php
public function post()
{
    return $this->belongsTo(Post::class);
}
```

---

3. Many to Many

Example: A Student belongs to many Courses

```php
// In Student.php
public function courses()
{
    return $this->belongsToMany(Course::class);
}

// In Course.php
public function students()
{
    return $this->belongsToMany(Student::class);
}
```

This needs a pivot table like `course_student`.
