Course code:CS-512 T

Total Credit:2 DSF-1A(2)

Course Title:C2. Advanced PHP

Class: BCSTY

Introduction to PHP

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.PHP is a recursive acronym for "PHP: Hypertext Preprocessor".PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.

It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.

PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.PHP is forgiving: PHP language tries to be as forgiving as possible.PHP Syntax is C-Like.

Common Uses of PHP PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them. The other uses of PHP are:PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.You add, delete, modify elements within your database thru PHP.Access cookies variables and set cookies.Using PHP, you can restrict users to access some pages of your website.It can encrypt data.

Advanced PHP

PHP is a server scripting language and a powerful tool for developing dynamic web applications Its formidability is why PHP Development skills are in demand,whether a WordPress or PHP site. It turns tables for web pages; the changes made to its newer versions have made user interaction easier and more viable. PHP is easier to learn,which makes it popular among seasoned developers. It powers more than 80% of published web pages today and is the go-to business solution.

PHP basics are easy to hone — but there are some advanced PHP concepts every

Unit I :Form Interaction

Form Interactions is the count of interactions that users have had with your web form. An interaction includes clicks, form entries, clicking a field to fill in information, and form abandonment. By tracking Form Interactions, you can learn where and why users abandon your forms, and how to modify your form to best capture your target audience.
Interactions count the number of times users perform a set of actions on your form, including clicks, submissions, characters entered into fields, and form abandonment. When you track this number, you can compare it with other Form Analytics metrics to gauge your form's performance. For example, Form Interactions can be tracked against
Form Submissions to judge trends as positive or negative. If Interactions increase but Submissions decrease, this means that the form is pushing away or not interesting users. If both Interactions and Submissions trend up, this can be taken as a positive trend.

Introduction to Web Form

web form (or HTML form) is a place where users enter data or personal information that's then sent to a server for processing. For example, users can share their name and email address to sign up for a newsletter or place an order What are web forms used for? Web forms are a great way to help you capture your website visitors' details – their name, email address, preferences, comments, and feedback. They are also a powerful means for website visitors to get in touch with a company, send information, place an order, send a request or an inquiry.

Forms are the basic interface between user and server. Form handling is the very basic and important feature of PHP. Using forms we can accept data from users and then we can handle the data using PHP. Data can be saved in any database server like MySql Post request is widely used to submit form that have large amount of data such as file.
upload, image upload, login form, registration form etc. The data passed through post request is not visible on the URL browser so it is secured. You can send large amount of data through post request.
Web forms, also known as HTML forms, are digital form interfaces on web pages that allow users to input data or interact with a website by filling out fields and submitting information.

Examples:

Contact Form., Payment Form, File Upload, Survey.

Processing Web Form

We will discuss how to process form in PHP. HTML forms are used to send the user information to the server and returns the result back to the browser. For example, if you want to get the details of visitors to your website, and send them good thoughts, you can collect the user information by means of form processing. Then, the information can be validated either at the client-side or on the server-side. The final result is sent to the client through the respective web browser. To create a HTML form, form tag should be used.

The PHP superglobals $_GET and $_POST are used to collect form-data.

Capturing form data

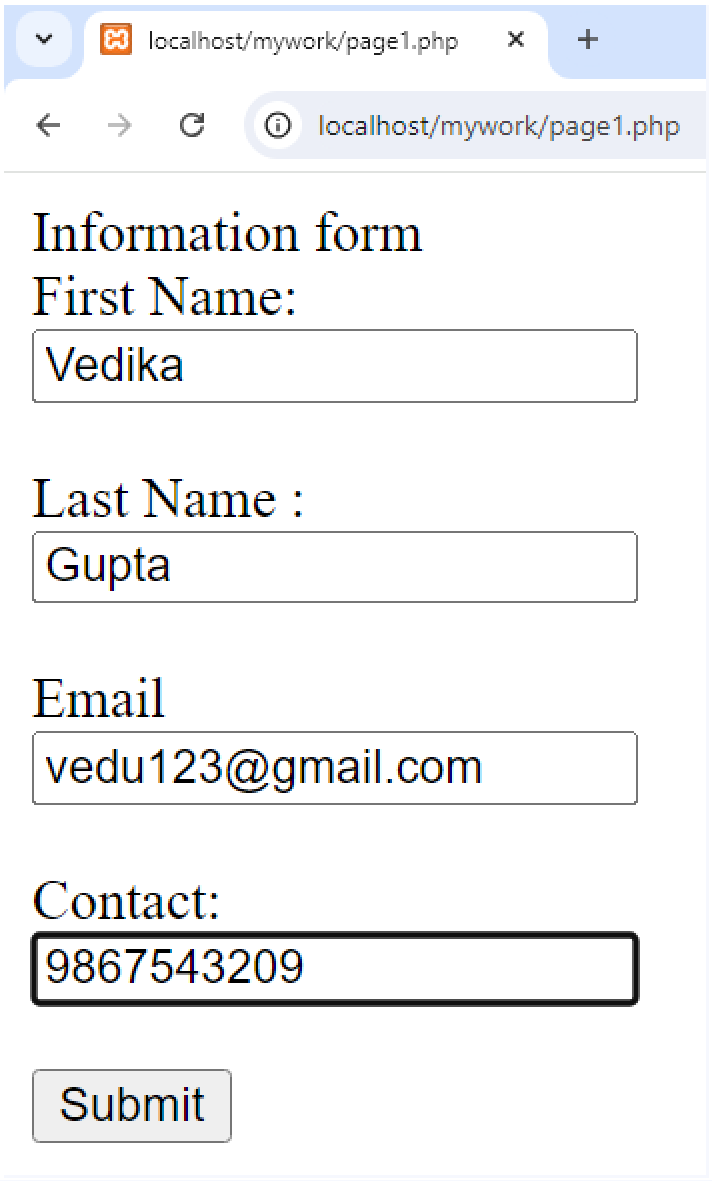
Capturing form data typically involves collecting user input from a web form and processing it for various purposes, such as saving it to a database, sending it via email, or using it within your application.Design your form using HTML. You can include various input fields such as text, email, password, radio buttons, checkboxes, and more.

Create Information form using HTML script

Page1.php

```
<!DOCTYPE html>
<html>
   <head><tittle>Information form</tittle></head>
   <body>
  <form action="page2.php" method="get">
           <lable for="fname">  First Name:</lable><br>
      <input type="text" name="fname" required><br><br>
      <lable for="fname">Last Name :</lable><br>
       <input type="text" name="lname" required><br><br>
        <lable for="fname">  Email</lable><br>
        <input type="email" name="email" required><br><br>
        <lable for="fname">  Contact:</lable><br>
        <input type="text" name="phone" required><br><br>
        <input type="submit" >

   </body>
</html>
```



Information form

First Name: Vedika

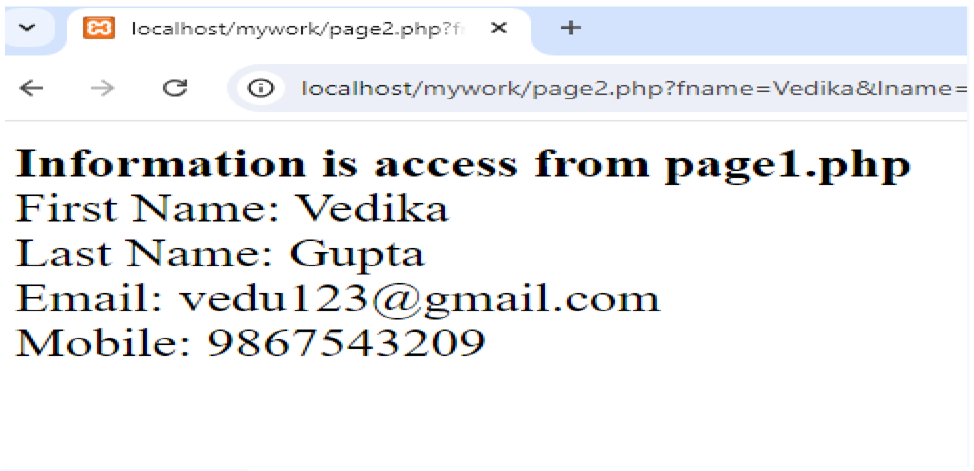
Last Name : Gupta

Email vedu123@gmail.com

Contact: 9867543209

The above information is passed or send to on page2.php using PHP Script

```
<?php
echo "<b>Information is access from page1.php </b><br>";
echo"First Name:".$_REQUEST['fname']."<br>";
echo"Last Name:".$_REQUEST['lname']."<br>";
echo"Email:".$_REQUEST['email']."<br>";
echo"Mobile:".$_REQUEST['phone']."<br>";
?>
```


localhost/mywork/page2.php?f
localhost/mywork/page2.php?fname=Vedika&lname=

```
Information is access from page1.php
First Name: Vedika
Last Name: Gupta
Email: vedu123@gmail.com
Mobile: 9867543209
```

Super Global Variables

These are specially-defined array variables in PHP that make it easy for you to get information about a request or its context. The superglobals are available throughout your script. These variables can be accessed from any function, class or any file without doing any special task such as declaring any global variable etc. They are mainly used to store and get information from one page to another etc in an application.

$GLOBALS, $_SERVER, $_REQUEST, $_GET, $_POST

$GLOBALS

$GLOBALS : It is a superglobal variable which is used to access global variables from anywhere in the PHP script. PHP stores all the global variables in array $GLOBALS[] where index holds the global variable name, which can be accessed.

Below program use of $GLOBALS in PHP:

```
<?php
$x = 300;
$y = 200;

function multiplication(){
   $GLOBALS['z'] = $GLOBALS['x'] * $GLOBALS['y'];
}

multiplication() ;
echo $z ;
?>
```

Output :

```
60000
```

In the above code two global variables are declared **$x** and **$y** which are assigned some value to them. Then a function **multiplication()** is defined to multiply the values of **$x** and **$y** and store in another variable **$z** defined in the **GLOBAL array**.

$_SERVER

$_SERVER : It is a PHP super global variable that stores the information about headers, paths and script locations. Some of these elements are used to get the information from the superglobal variable $_SERVER. Below program illustrates the use of $_SERVER in PHP:

$_SERVER['PHP_SELF'] --→ Returns the filename of the currently executing script

```
<?php
        echo $_SERVER['PHP_SELF'];
?>
```

$_SERVER['SERVER_NAME'] -→ Returns the name of the host server

```
<?php
        echo $_SERVER['SERVER_NAME'];
?>
```

$_SERVER['HTTP_HOST'] → Returns the name of the host server

```
<?php
      echo $_SERVER['HTTP_HOST'];
?>
```

$_SERVER['SCRIPT_NAME'] → Returns the name of the host server

```
<?php
      echo  $_SERVER['SCRIPT_NAME'];
?>
```

$_SERVER['REQUEST_TIME'] → Returns the timestamp of the start of the request (such as 1377687496)

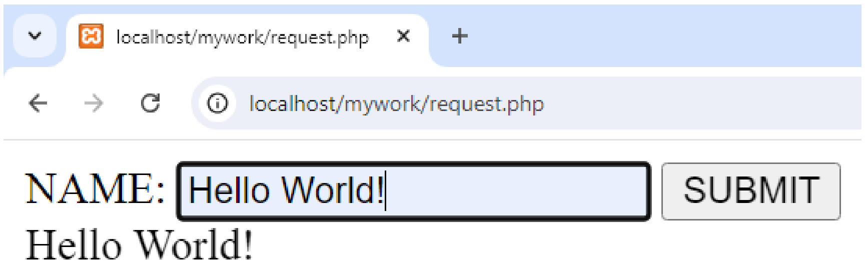
```
<?php
         echo $_SERVER[‘REQUEST_TIME’];
?>
```

$_REQUEST :

It is a superglobal variable which is used to collect the data after submitting a HTML form. $_REQUEST is not used mostly, because $_POST and $_GET perform the same task and are widely used.
Below is the HTML and PHP code to explain how $_REQUEST works:

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

```
<form method="post" target="_SELF">
NAME: <input type="text" name="fname">
<button type="submit">SUBMIT</button>
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $name = htmlspecialchars($_REQUEST['fname']);
   if(empty($name)){
            echo "Name is empty";
   } else {
            echo $name;
   }
}
?>
</body>
</html>
Output:
```



```
Hello World!
```

In the above code we have created a form that takes the name as input from the user and prints it's name on clicking of submit button. We transport the data accepted in the form to the same page using **target="_SELF"** element as specified in the action attribute, because we manipulate the data in the same page using the PHP code. The data is retrieved using the $_REQUEST superglobal array variable

The Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. HTTP works as a request-response protocol between a client and server. A web browser may be the client, and an application on a computer that hosts a website may be the server. A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.
There are 2 HTTP request methods:

- **GET:** Requests data from a specified resource.
- **POST:** Submits data to be processed to a specified resource.

POST Method:

In the **POST method,** the data is sent to the server as a package in a separate communication with the processing script. Data sent through the POST method will not be visible in the URL.

Advantages:

- It is more secure than GET because user-entered information is never visible in the URL query string or in the server logs.
- There is a much larger limit on the amount of data that can be passed and one can send text data as well as binary data (uploading a file) using POST.

Disadvantages:

- Since the data sent by the POST method is not visible in the URL, so it is not possible to bookmark the page with a specific query.
- POST requests are never cached
- POST requests do not remain in the browser history.

$_POST :

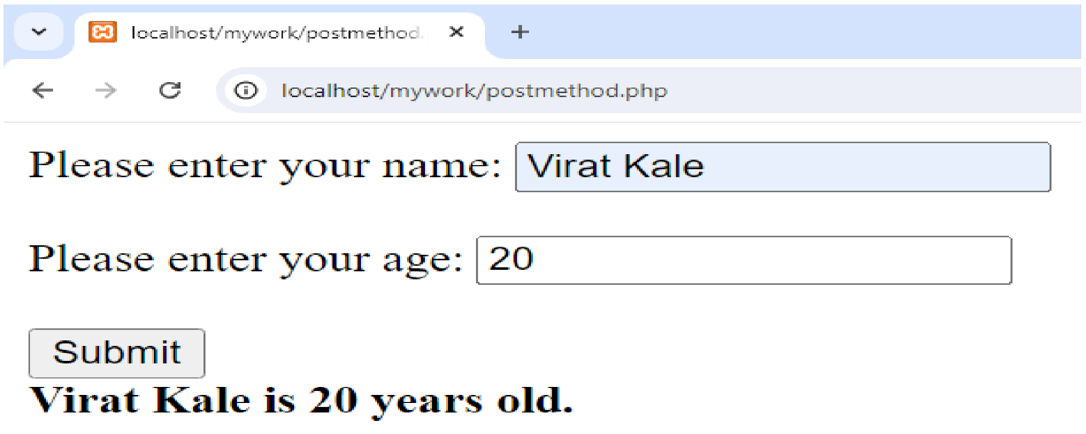
It is a **super global variable** used to collect data from the HTML form after submitting it. When form uses method post to transfer data, the data is not visible in the query string, because of which security levels are maintained in this method.

The following program is using $_POST variable

```
<!DOCTYPE html>
<html>
<body>
  <form method="post" action=" ">
 <label for="name">Please enter your name: </label>
 <input name="name" type="text"><br><br>
 <label for="age">Please enter your age: </label>
 <input name="age" type="text"><br><br>
 <input type="submit" value="Submit">
</form>
<?php
$nm=$_POST['name'];
$age=$_POST['age'];
echo "<strong>".$nm." is $age years old.</strong>";
?>
```

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

Outputs:



Please enter your name: Virat Kale

Please enter your age: 20

Virat Kale is 20 years old.

GET Method: In the **GET method,** the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands (&). In general, a URL with GET data will look like this:

Advantages:

- Since the data sent by the GET method are displayed in the URL, it is possible to bookmark the page with specific query string values.
- GET requests can be cached and GET requests to remain in the browser history.
- GET requests can be bookmarked.

Disadvantages:

- The GET method is not suitable for passing sensitive information such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.
- Because the GET method assigns data to a server environment variable, the length of the URL is limited. So, there is a limitation for the total data to be sent.

$_GET

In the PHP file we can use the $_**GET super global variable** to collect the value of the query string. $_GET contains an array of variables received via the HTTP GET method.

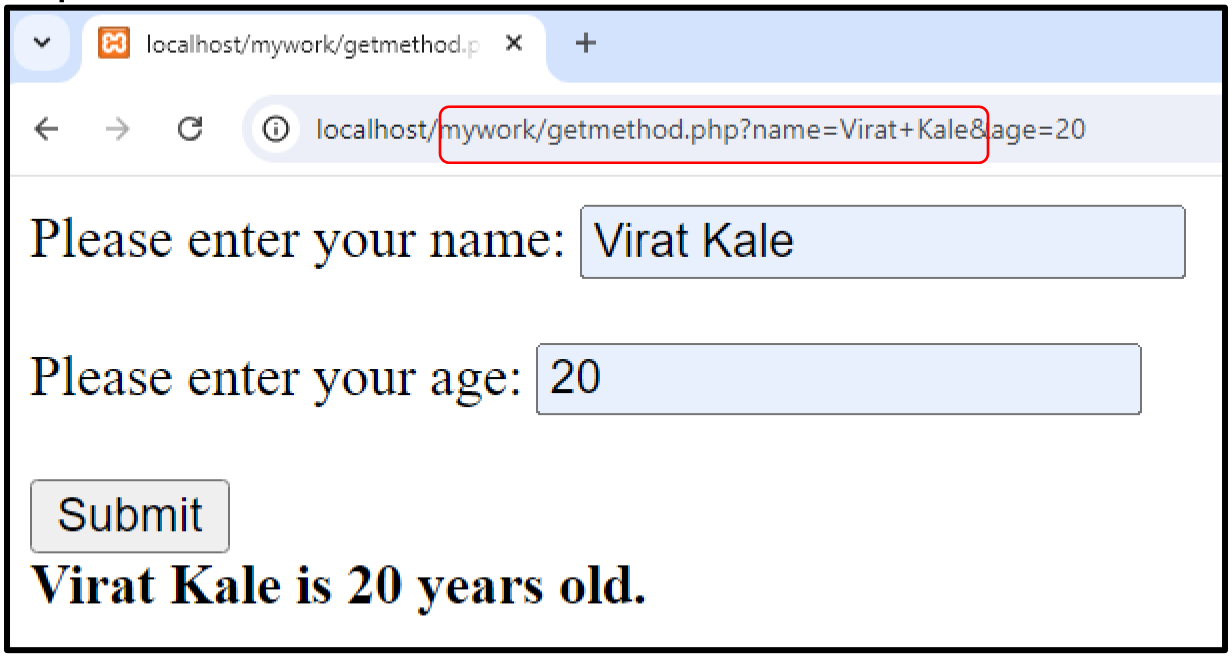
The following program is using $_GET variable

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

<form method="get" target="_SELF">
```

```
<label for="name">Please enter your name: </label>
<input name="name" type="text"><br><br>
 <label for="age">Please enter your age: </label>
 <input name="age" type="text"><br><br>
 <input type="submit" value="Submit">
</form>
<?php
$nm=$_GET['name'];
$age=$_GET['age'];
echo "<strong>".$nm." is $age years old.</strong>";
?>
</body>
</html>
```

Outputs:



```
localhost/mywork/getmethod.php?name=Virat+Kale&age=20
```

Please enter your name: Virat Kale

Please enter your age: 20

Submit

Virat Kale is 20 years old.

Working with Number

PHP Numbers

There are three main numeric types in PHP:

Integer, Float, Number, Strings ,var_dump()

To verify the type of any object in PHP, use the var_dump() function:

```
 $a = 5;
 $b = 5.34;
 $c = "25";

 var_dump($a);
 var_dump($b);
 var_dump($c);
```
Output
```
int(5)
float(5.34)
string(2) "25"
```

Integer:

PHP Integers

2, 256, -256, 10358, -179567 are all integers.

An integer is a number without any decimal part.

An integer data type is a non-decimal number between -2147483648 and 2147483647 in 32 bit systems, and between -9223372036854775808 and 9223372036854775807 in 64 bit systems. A value greater (or lower) than this, will be stored as float, because it exceeds the limit of an integer.

Note: Another important thing to know is that even if 4 * 2.5 is 10, the result is stored as float, because one of the operands is a float (2.5).

Here are some rules for integers:

An integer must have at least one digit
An integer must NOT have a decimal point
An integer can be either positive or negative
Integers can be specified in three formats: decimal (base 10), hexadecimal (base 16 - prefixed with 0x), octal (base 8 - prefixed with 0) or binary (base 2 - prefixed with 0b)
PHP has the following predefined constants for integers:

PHP_INT_MAX - The largest integer supported
PHP_INT_MIN - The smallest integer supported
PHP_INT_SIZE - The size of an integer in bytes
PHP has the following functions to check if the type of a variable is integer:

is_int()
is_integer() - alias of is_int()
is_long() - alias of is_int()

Example

Check if the type of a variable is integer:

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

<?php
// Check if the type of a variable is integer
$x = 5985;
var_dump(is_int($x));

echo "<br>";

// Check again...
$x = 59.85;
var_dump(is_int($x));
?>

</body>
</html>
```

Output:

```
bool(true)
bool(false)
```

Floats

A float is a number with a decimal point or a number in exponential form.

2.0, 256.4, 10.358, 7.64E+5, 5.56E-5 are all floats.

The float data type can commonly store a value up to 1.7976931348623E+308 (platform dependent), and have a maximum precision of 14 digits.

PHP has the following predefined constants for floats (from PHP 7.2):

PHP_FLOAT_MAX - The largest representable floating point number
PHP_FLOAT_MIN - The smallest representable positive floating point number
PHP_FLOAT_DIG - The number of decimal digits that can be rounded into a float and back without precision loss
PHP_FLOAT_EPSILON - The smallest representable positive number x, so that x + 1.0 != 1.0
PHP has the following functions to check if the type of a variable is float:

is_float()
is_double() - alias of is_float()
Example
Check if the type of a variable is float:

```
<?php
// Check if the type of a variable is float
$x = 10.365;
var_dump(is_float($x));
?>
```
Output: bool(true)

Infinity

A numeric value that is larger than PHP_FLOAT_MAX is considered infinite.

PHP has the following functions to check if a numeric value is finite or infinite:

is_finite()
is_infinite()
However, the PHP var_dump() function returns the data type and value:

Example
Check if a numeric value is finite or infinite:
```
<?php
$x = 1.9e411


  echo var_dump(is_finite($x));

?>
```
Output: float(INF)

Numerical Strings

The PHP is_numeric() function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

Example:
```
<?php
// Check if the variable is numeric
$x = 5985;
var_dump(is_numeric($x));

echo "<br>";

$x = "5985";
var_dump(is_numeric($x));

echo "<br>";

$x = "59.85" + 100;
```

```
var_dump(is_numeric($x));

echo "<br>";

$x = "Hello";
var_dump(is_numeric($x));
?>
```

Output:

```
bool(true)
bool(true)
bool(true)
bool(false)
```

PHP Casting Strings and Floats to Integers

Sometimes you need to cast a numerical value into another data type.
The (int), (integer), and intval() functions are often used to convert a value to an integer.

```
<?php
// Cast float to int
$n = 43667865.453;
$int_cast = (int)$n;
echo $int_cast;

echo "<br>";

// Cast string to int
$x = " 768955355.4767";
$int_cast = (int)$x;
echo $int_cast;
?>
```

Output:

```
 43667865
             768955355
```

Change Data Type
Casting in PHP is done with these statements:
(string) - Converts to data type String

```
a = 5;       // Integer
$b = 5.34;    // Float
$c = "hello"; // String
$d = true;    // Boolean
$e = NULL;
$a = (string) $a;
$b = (string) $b;
$c = (string) $c;
$d = (string) $d;
```

```
$e = (string) $e;

//To verify the type of any object in PHP, use the var_dump() function:
var_dump($a);
var_dump($b);
var_dump($c);
var_dump($d);
var_dump($e);
?>
```

Outputs: string(1) "5" string(4) "5.34" string(5) "hello" string(1) "1" string(0)

(int) - Converts to data type Integer

```
 <?php
          $a=98.99;//float Value
                $b="545.633"; //String value
                $a = (int) $a;
          echo "value  of a=98.99 now" ." ".$a;
          echo "<br>";
          $b= (int) $b;
                echo "value of b=\"545.633\":"." ".$b;
          ?>
```

Outputs:

```
value of a=98.99 now 98
value of b="545.633": 545
```

(float) - Converts to data type Float

Eg:
```
<?php
        $b="7556.89"; //String value
        $b= (float) $b;
             echo "value of b=\"$b\":"." ".$b;
         ?>
```

Outputs:

```
value of b="7556.89": 7556.89
```

(unset) - Converts to data type NULL

```
<?php
                $a=564645;
                $b="Hello"; //String value
                echo "value  of $a now :" ." ". (unset) $a;
                echo "<br>";
                $b= (unset) $b;
                echo "value of b=\"$b\":"." ".$b."<BR>";
                ?>
```

Outputs:

```
value of a=564645 now:
value of b="":
```

pi() Function

The pi() function returns the value of PI:

```
<?php
echo(pi());
?>
```

Output:

```
3.1415926535898
```

min() and max() Functions

The min() and max() functions can be used to find the lowest or highest value in a list of arguments

```
<?php
echo(min(0, 150, 30, 20, -8, -200) . "<br>");
echo(max(0, 150, 30, 20, -8, -200));
?>
```

Output:

```
-200
150
```

abs() Function

The abs() function returns the absolute (positive) value of a number:

Example:

```
<?php
echo(abs(-6.7));
?>//Output: 6.7
```

round() Function

The round() function rounds a floating-point number to its nearest integer:

Example:

```
<?php
echo(round(0.60) . "<br>");
echo(round(0.50) . "<br>");
echo(round(0.49) . "<br>");
echo(round(-4.40) . "<br>");
echo(round(-4.60));
?>//Outputs : 1 1 0 4 -5
```

Random Numbers

The rand() function generates a random number:

Example:

```
<?php
<?php
echo(rand(10, 100));
?>//Outputs :   97
```

Working with String

A string is a sequence of characters, like "Hello world!

```
<?php
echo "Hello";
print "Hello";
?>
```

You can use double or single quotes, but you should be aware of the differences between the two.
Double quoted strings perform action on special characters.
E.g. when there is a variable in the string, it returns the *value* of the variable:

```
<?php
$x = "John";
echo "Hello $x";
?>
```

String Length

The PHP strlen() function returns the length of a string.
Example:

```
<?php
echo strlen("Hello world!");
?> //Outputs :12
```

Word Count

The PHP str_word_count() function counts the number of words in a string.
Count the number of word in the string "Hello world!":
Example:

```
<?php
echo str_word_count("Hello world!");
?>// Outputs: 2
```

Search For Text Within a String

The PHP strpos() function searches for a specific text within a string.
If a match is found, the function returns the character position of the first match.
If no match is found, it will return FALSE.
Search for the text "world" in the string "Hello world!":
Example:

```
<?php
echo strpos("Hello world!", "world");
?> //Output: 6
```

Modify Strings

Upper Case

The strtoupper() function returns the string in upper case:

Example:

```
$x = "Hello World!";
echo strtoupper($x);/
```

Lower Case

The strtolower() function returns the string in lower case:

Example:

```
$x = "Hello World!";
echo strtolower($x);
```

Replace String

The PHP str_replace() function replaces some characters with some other characters in a string.

Replace the text "World" with "Dolly":

Example:

```
$x = "Hello World!";
echo str_replace("World", "Dolly", $x);
```

Reverse a String

The PHP strrev() function reverses a string.

Reverse the string "Hello World!":

Example:

```
$x = "Hello World!";
echo strrev($x);
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

The trim()

removes any whitespace from the beginning or the end:

Example:

```
$x = " Hello World! ";
echo trim($x);
```

Learn more in our trim() Function Reference.

Convert String into Array

The PHP explode() function splits a string into an array.

The first parameter of the explode() function represents the "separator". The "separator" specifies where to split the string.

Note: The separator is required.

Example
Split the string into an array. Use the space character as separator:

```
<?php
$x = "Hello World!";
$y = explode(" ", $x);

//Use the print_r() function to display the result:
print_r($y);
?>
```

Output:

```
Array ( [0] => Hello [1] => World! )
```

String Concatenation

To concatenate, or combine, two strings you can use the . operator:
Example:

```
<?php
$x = "Hello";
$y = "World";

$z = $x . $y;
echo $z;
?>//Output: HelloWorld
```

The result of the example above is HelloWorld, without a space between the two words.
You can add a space character like this:
Example:

```
<?php
$x = "Hello";
$y = "World";
$z = $x . " " . $y;
echo $z;
?>
```

An easier and better way is by using the **power of double quotes.**
By surrounding the two variables in double quotes with a white space between them, the white space will also be present in the result:
Example:

```
<?php
$x = "Hello";
$y = "World";
$z = "$x $y";
echo $z;
?>//Output: Hello World
```

Slicing Strings

Slicing

You can return a range of characters by using the substr() function.
Specify the start index and the number of characters you want to return.
Example :
Start the slice at index 6 and end the slice 5 positions later:

```
<?php
$x = "Hello World!";
echo substr($x, 6, 5);
?>//Output: World
```

Note The first character has index 0.

Slice to the End

By leaving out the *length* parameter, the range will go to the end:
Example:
Start the slice at index 6 and go all the way to the end:

```
<?php
$x = "Hello World!";
echo substr($x, 6);
?>//Output: World!
```

Slice *From* the End

Use negative indexes to start the slice from the end of the string:
Example
Get the 3 characters, starting from the "o" in world (index -5):

```
<?php
$x = "Hello World!";
echo substr($x, -5, 3);
?>Output: orl
```

Note The last character has index -1.

Negative Length

Use negative *length* to specify how many characters to omit, starting from the end of the string:
Example:
From the string "Hi, how are you?", get the characters starting from index 5, and continue until you reach the 3. character from the end (index -3).
Should end up with "ow are y":

```
<?php
$x = "Hi, how are you?";
echo substr($x, 5, -3);
?> //Output: ow are y
```

Escape Character

To insert characters that are illegal in a string, use an escape character.
An escape character is a backslash \ followed by the character you want to insert.
An example of an illegal character is a double quote inside a string that is surrounded by double quotes:
Example:
$x = "We are the so-called "Vikings" from the north.";
To fix this problem, use the escape character \":
Example:
$x = "We are the so-called \"Vikings\" from the north.";

In PHP, you can work with dates and times using the date() function, DateTime class, and several other related functions. Here's a quick overview of the most

Code	Result
\'	Single Quote
\"	Double Quote
\$	PHP variables
\n	New Line
\r	Carriage Return
\t	Tab

commonly used methods:
Date And Time Function
1. date() Function
The date() function formats a local date and time according to a specified format.
Example:
echo date('Y-m-d H:i:s'); // **Outputs**: 2024-08-18 12:34:56
- Y: A full numeric representation of a year, 4 digits.
- m: Numeric representation of a month, with leading zeros.
- d: Day of the month, 2 digits with leading zeros.
- H: 24-hour format of an hour with leading zeros.
- i: Minutes with leading zeros.
- s: Seconds with leading zeros.
- l : character represent day name

2. time() Function
The time() function returns the current time as a Unix timestamp (the number of seconds since January 1, 1970).
Example:

```
echo time();  // Outputs: 1728909296
```

3. mktime() Function

The mktime() function returns the Unix timestamp for a date.

Example:

```
$timestamp = mktime(14, 30, 0, 8, 18, 2024);
echo date('Y-m-d H:i:s', $timestamp);  // Outputs: 2024-08-18 14:30:00
```

4. strtotime() Function

The strtotime() function converts a textual datetime description into a Unix timestamp.

Example:

```
$timestamp = strtotime('next Monday');
echo date('Y-m-d', $timestamp);  // Outputs: the date of the next Monday
```

5. DateTime Class

The DateTime class provides a more object-oriented approach to date and time handling.

Example:

```
$date = new DateTime();
echo $date->format('Y-m-d H:i:s');  // Outputs: current date and time

// Modifying a date
$date->modify('+1 day');
echo $date->format('Y-m-d H:i:s');  // Outputs: date and time of the next day

// Create DateTime from a specific date
$date = new DateTime('2024-08-18 14:30:00');
echo $date->format('Y-m-d H:i:s');  // Outputs: 2024-08-18 14:30:00
```

6. DateInterval and DatePeriod

These classes allow you to perform operations with intervals (like adding/subtracting days, months, etc.) and periods.

```
$date = new DateTime();
$date->add(new DateInterval('P2D'));  // Adds 2 days
echo $date->format('Y-m-d H:i:s');

$period = new DatePeriod(
    new DateTime('2024-08-01'),
    new DateInterval('P1D'),
    new DateTime('2024-08-05')
    foreach ($period as $dt) {
    echo $dt->format('Y-m-d') . "\n";  // Outputs dates from 2024-08-01 to 2024-08-
04
}
```

These functions and classes give you a wide range of tools to work with dates and times in PHP.

Sub :Advanced PHP

UNIT II Validation

Form Validation in PHP

An HTML form contains various input fields such as text box, checkbox, radio buttons, submit button, and checklist, etc. These input fields need to be validated, which ensures that the user has entered information in all the required fields and also validates that the information provided by the user is valid and correct.
There is no guarantee that the information provided by the user is always correct. PHP validates the data at the server-side, which is submitted by HTML form. You need to validate a few things:

Empty String, Validate String, Validate Numbers, Validate Email, Validate URL, Input length

Interaction with HTML Form

Creating forms in HTML involves using various elements and attributes to collect user input.
Here's a basic guide to help you understand and create interactive forms.

Basic Form Structure

A basic HTML form is defined using the <form> element. Inside the form, you can use various input elements like <input>, <textarea>, <select>, and <button>.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple Form</title>
</head>
<body>
  <form action="/submit" method="post">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
    <br><br>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <br><br>

    <label for="message">Message:</label>
    <textarea id="message" name="message" rows="4" cols="50" required></textarea>
    <br><br>

    <label for="gender">Gender:</label>
    <input type="radio" id="male" name="gender" value="male">
    <label for="male">Male</label>
    <input type="radio" id="female" name="gender" value="female">
    <label for="female">Female</label>
    <br><br>

    <label for="newsletter">Subscribe to newsletter:</label>
    <input type="checkbox" id="newsletter" name="newsletter" value="yes">
    <br><br>

    <label for="country">Country:</label>
    <select id="country" name="country">
```

```
            <option value="usa">United States</option>
            <option value="canada">Canada</option>
            <option value="uk">United Kingdom</option>
        </select>
        <br><br>

        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

Key Form Elements

1. **<form>**: Encapsulates all the form elements.
   - **action**: URL where the form data will be sent when submitted.
   - **method**: HTTP method used for sending the form data (get or post).
2. **<label>**: Provides a label for form elements. It's linked to an input element by using the for attribute.
3. **<input>**: Various types are used for different kinds of data:
   - **type="text"**: Single-line text input.
   - **type="email"**: Email input with validation.
   - **type="radio"**: Radio buttons for selecting one option from a group.
   - **type="checkbox"**: Checkbox for binary choices.
   - **type="submit"**: Submit button to send form data.
4. **<textarea>**: Multi-line text input for longer messages.
5. **<select>** and **<option>**: Dropdown list for selecting one option from multiple choices.
6. **<button>**: Can be used to submit the form or perform other actions.

Form Attributes

- **required**: Ensures that a field must be filled out before submitting.
- **placeholder**: Provides a hint within the input field.

Handling Form Data

Forms can be processed on the server side. For example, if you specify action="/submit" and method="post", when the user submits the form, the data will be sent to /submit using the POST method.

Validating HTML forms can be achieved using a combination of HTML attributes, CSS, and JavaScript. HTML5 introduced several new attributes and input types that make client-side validation easier and more powerful. Here’s a breakdown of how you can validate HTML forms:

HTML Form Validation

HTML5 provides built-in validation attributes that you can use to enforce rules on form inputs:

- **required**: Specifies that the field must be filled out before submitting the form.
  ```
  <input type="text" name="username" required>
  ```
- **minlength** and **maxlength**: Define the minimum and maximum length of input.
  ```
  <input type="text" name="username" minlength="3" maxlength="20" required>
  ```
- **pattern**: Uses a regular expression to enforce specific input patterns.
  ```
  <input type="text" name="zipcode" pattern="\d{5}" title="Five digit zip code" required>
  ```
- **type**: Some input types provide automatic validation, such as email, number, and url.
  ```
  <input type="email" name="email" required>
  <input type="number" name="age" min="1" max="120">
  ```
- **min** and **max**: Used with number and date inputs to define acceptable ranges.
  ```
  <input type="number" name="quantity" min="1" max="100" required>
  <input type="date" name="birthdate" min="1900-01-01" max="2024-01-01">
  ```
- **step**: Defines the legal number intervals for number and range inputs.

```
      <input type="number" name="amount" step="0.01" required>
```

Validating HtML Form form: name, email, number password using Javascript

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

    <title>Form Validation</title>
</head>
<body >
<script>
    function data()
    {
   var a=document.getElementById("1").value;
   var b=document.getElementById("2").value;
   var c=document.getElementById("3").value;
   var d=document.getElementById("4").value;
   var e=document.getElementById("5").value;
   var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
   if(a==""||b==""||c==""||d==""||e=="")// Validation for empty field
   {
    alert("All field are mendatory");
    return false;
   }
   else if (!/^[a-zA-Z\s]+$/.test(a)) {
    error = "Name can only contain letters and spaces.";
    alert("Only letters are allowed, please enter valid name");
    return false;
   }

  else if(b.length<10||b.length>10)  //Validation for contact no should  be 10 digit
   {
    alert(" Contact Number should be 10 digit plase enter valid number");
    //document.myForm.phone.focus();
    return false;
   }
   else if(isNaN(b))//contact number should be only digit
   {
    alert("Only Numbers are allowed");
    return false;
   }

   else if (!emailPattern.test(c)) {  //Email validation
                error = "Please enter a valid email address.";
                alert("Please enter a valid email address.");
    return false;
           }

   else if(d!=e)   //Password validation
   {
    alert("Please input same password");
    return false;
   }  else {
    true;
   }
   }

</script>
<form name="myForm" action="javaform2.php" onsubmit=" return data()"  >
      <div>
       <lable for="name">Name:</lable><br>
        <input type="text" name="name" id="1"><br><br>
        </div>


        <div>
        <lable for="phone">Mobile:</lable><br>
        <input type="text" name="phone" id="2" ><br><br>
        </div>
        <div>
        <lable for="email">Email:</lable><br>
        <input type="email" name="email" id="3" ><br><br>
        </div>
        <div>
        <lable for="pass">Password:</lable><br>
        <input type="password" name="password" id="4" ><br><br>
        </div>
        <div>
        <div>
        <lable for="password">Confirm Password:</lable><br>
        <input type="password" name="password" id="5" ><br><br>
        </div>
        <div>
        <div>
        <lable for="submit">Please submit Information</lable><br>
        <input type="submit" name="submit">
        </div>
        <?php

      ?>

    </body>
    </html>
```

Input validation

Input validation is the process of ensuring that the data entered into a form or application meets certain criteria before it is processed or saved. This process is crucial for maintaining data integrity, preventing errors, and ensuring security. Here's a detailed overview of input validation:

1. **Purpose of Input Validation**

- **Data Integrity**: Ensures that the data entered is accurate and in the correct format.
- **Security**: Prevents malicious inputs that could lead to security vulnerabilities like SQL injection or XSS (Cross-Site Scripting).
- **User Experience**: Provides immediate feedback to users if their input is incorrect, helping them correct errors before submission.

2. **Types of Input Validation**

1. **Client-Side Validation**
   - **HTML5 Attributes**: Modern HTML5 provides built-in attributes for basic validation.

     ```
     <input type="email" name="email" required>
     <input type="number" name="age" min="1" max="120">
     ```

   1.
   - **JavaScript**: Custom validation logic to handle more complex scenarios.

     ```
     document.getElementById('myForm').addEventListener('submit', function(event) {
        let isValid = true;
        const email = document.getElementById('email').value;
        if (!email.includes('@')) {
            document.getElementById('emailError').textContent = 'Invalid email
     address.';
            isValid = false;
        }
        if (!isValid) {
            event.preventDefault();
        }
     });
     ```

2. **Server-Side Validation**
   - **Server-Side Scripting**: Languages like PHP, Python, Ruby, and Node.js can be used to validate data after submission.

     ```
     if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $errors[] = "Invalid email format";
     }
     ```

   - **Database Constraints**: Enforce data integrity at the database level, such as unique keys, foreign keys, and data types.

3. **Validation Techniques**

- **Type Checking**: Ensures that the data is of the expected type (e.g., number, string, date).

  ```
  <input type="number" name="age" min="1">
  ```

- **Format Checking**: Ensures that data follows a specific format (e.g., email, phone number).

  ```
  <input type="email" name="email">
  ```

- **Range Checking**: Ensures that data falls within a specified range (e.g., age, quantity).

```
<input type="number" name="quantity" min="1" max="100">
```

- **Length Checking**: Ensures that the length of the input meets certain criteria (e.g., password length).

```
<input type="text" name="username" minlength="3" maxlength="15">
```

- **Presence Checking**: Ensures that the field is not empty.

```
<input type="text" name="name" required>
```

Exception and Error Handling
Exception

In PHP, exceptions are used to handle errors and exceptional conditions in a more controlled and flexible way compared to traditional error handling methods. Exceptions allow you to manage errors using try, catch, and finally blocks, which makes your code cleaner and more maintainable.

Basic Concepts of Exceptions in PHP

1. **Exception Class**: The base class for all exceptions is Exception. You can extend this class to create custom exceptions.
2. **Throwing Exceptions**: Use the throw keyword to raise an exception when an error condition is detected.
3. **Catching Exceptions**: Use the try and catch blocks to handle exceptions. The catch block will execute if an exception is thrown within the try block.
4. **Finally Block**: (Optional) The finally block, if present, will execute after the try and catch blocks, regardless of whether an exception was thrown or not.

Basic Example

Here's a basic example demonstrating how to throw and catch exceptions in PHP:

```
<?php
class CustomException extends Exception {}

function divide($numerator, $denominator) {
   if ($denominator == 0) {
      throw new CustomException("Cannot divide by zero.");
   }
   return $numerator / $denominator;
}

try {
   echo divide(10, 2); // This will work
   echo divide(10, 0); // This will throw an exception
} catch (CustomException $e) {
   echo "Caught exception: " . $e->getMessage();
} finally {
   echo "Execution complete.";
}
?>
```
Explanation

1. **Custom Exception**: CustomException extends the base Exception class. This allows you to create exceptions with custom behavior or messages.
2. **Throwing an Exception**: The divide function throws a CustomException if the denominator is zero.
3. **Catching an Exception**: The catch block catches the CustomException and handles it by displaying an error message.
4. **Finally Block**: The finally block will execute regardless of whether an exception was thrown or not. It's useful for cleanup tasks like closing files or database connections.

Built-In Exception Handling

PHP provides several built-in exceptions for common error scenarios:

- **ErrorException**: Converts PHP errors into exceptions. This is useful for handling legacy code or integrating error handling with exceptions.

  ```
  set_error_handler(function($errno, $errstr, $errfile, $errline) {
     throw new ErrorException($errstr, 0, $errno, $errfile, $errline);
  });

  try {
     echo $undefinedVariable; // This will cause an error
  } catch (ErrorException $e) {
     echo "Caught exception: " . $e->getMessage();
  }
  ```

- **PDOException**: Used with the PDO (PHP Data Objects) extension to handle database-related errors.

  ```
  try {
     $pdo = new PDO('mysql:host=localhost;dbname=test', 'user', 'password');
  } catch (PDOException $e) {
     echo "Connection failed: " . $e->getMessage();
  }
  ```

Creating Custom Exceptions

You can create your own exception classes to handle specific error conditions in your application:

```
<?php
class InvalidUserInputException extends Exception {}

function processUserInput($input) {
   if (empty($input)) {
      throw new InvalidUserInputException("User input cannot be empty.");
```

```
    }
    // Process input
}

try {
    processUserInput(""); // This will throw an exception
} catch (InvalidUserInputException $e) {
    echo "User input error: " . $e->getMessage();
}
?>
```

Exception Methods

Here are some useful methods provided by the Exception class:

- **getMessage()**: Returns the exception message.
- **getCode()**: Returns the exception code.
- **getFile()**: Returns the filename where the exception was created.
- **getLine()**: Returns the line number where the exception was created.

Summary

- **Exception Handling**: Use try, catch, and finally blocks to handle exceptions and manage error scenarios in a controlled manner.
- **Custom Exceptions**: Extend the Exception class to create custom exceptions tailored to your application's needs.
- **Built-in Exceptions**: Utilize PHP's built-in exceptions for common error handling tasks, such as database interactions and error-to-exception conversion.

Error handling

Error handling in PHP is essential for managing and responding to unexpected conditions in your code. PHP provides several mechanisms for error handling, including built-in error reporting, custom error handling functions, and exception handling. Here's a comprehensive guide to understanding and implementing error handling in PHP:

1. **Error Reporting**

PHP's error reporting settings control which errors are reported and displayed. You can configure error reporting in your PHP scripts or in the php.ini configuration file.

Configuring Error Reporting in php.ini

In the php.ini file, you can set the following directives:

```
error_reporting = E_ALL
display_errors = On
```

- **error_reporting**: Defines the types of errors to report. E_ALL reports all errors.

- **display_errors**: Determines whether errors should be displayed on the screen. Set to On for development; Off for production.

Configuring Error Reporting in PHP Script

You can configure error reporting directly in your PHP script using the error_reporting() and ini_set() functions:

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', '1');
?>
```

2. Custom Error Handling

PHP allows you to define custom error handlers using the set_error_handler() function. This function lets you specify a custom callback function to handle errors.

Example of Custom Error Handler

```
<?php
function customErrorHandler($errno, $errstr, $errfile, $errline) {
   echo "Error [$errno]: $errstr in $errfile on line $errline";
   // Log the error to a file
   error_log("Error [$errno]: $errstr in $errfile on line $errline", 3, 'errors.log');
}

// Set the custom error handler
set_error_handler('customErrorHandler');

// Trigger an error
echo $undefinedVariable;
?>
```

3. **Error Handling with Exceptions**

Exceptions provide a more flexible way to handle errors compared to traditional error handling. PHP uses try, catch, and finally blocks to manage exceptions.

Example of Exception Handling

```
<?php
class CustomException extends Exception {}

function riskyOperation() {
   throw new CustomException("Something went wrong!");
}

try {
   riskyOperation();
} catch (CustomException $e) {
   echo "Caught exception: " . $e->getMessage();
} finally {
    echo "This block always executes.";
}
?>
```

4. **Error Logging**

Error logging allows you to record errors for later review. You can log errors to a file or a database.

5. **Handling Different Error Types**

PHP defines several error types, each represented by an error number:

- **E_ERROR**: Fatal runtime errors.
- **E_WARNING**: Runtime warnings (non-fatal errors).
- **E_PARSE**: Compile-time parse errors.
- **E_NOTICE**: Runtime notices (often indicate potential problems).
- **E_DEPRECATED**: Warnings about deprecated features.
- **E_USER_ERROR**, **E_USER_WARNING**, **E_USER_NOTICE**: User-generated errors.

Example of Handling Specific Error Types

```
<?php
function customErrorHandler($errno, $errstr, $errfile, $errline) {
    switch ($errno) {
        case E_ERROR:
            echo "Fatal error: $errstr";
            break;
        case E_WARNING:
            echo "Warning: $errstr";
            break;
        default:
            echo "Error: $errstr";
            break;
    }
    // Log the error to a file
    error_log("Error [$errno]: $errstr in $errfile on line $errline", 3, 'errors.log');
}

set_error_handler('customErrorHandler');
?>
```

6. **Handling Errors in Object-Oriented PHP**

You can also use exceptions in object-oriented PHP to handle errors:

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

    public function setName($name) {
        if (empty($name)) {
            throw new InvalidArgumentException("Name cannot be empty.");
        }
        $this->name = $name;
    }
}

try {
    $user = new User();
    $user->setName(""); // This will throw an exception
} catch (InvalidArgumentException $e) {
    echo "Caught exception: " . $e->getMessage();
}
?>
```

Summary

- **Error Reporting**: Configure which errors are reported and displayed using php.ini or script settings.
- **Custom Error Handling**: Use set_error_handler() to define how errors are handled.
- **Exception Handling**: Use try, catch, and finally blocks to manage exceptions.
- **Error Logging**: Log errors to files or other storage systems for later analysis.
- **Error Types**: Handle specific types of errors based on their severity and nature

Cookies Handling in PHP

A **cookies** in PHP is a small piece of data stored on the user's computer by the web browser. It is typically used to store user-specific information, such as login sessions, preferences, or any other data that needs to persist between different pages or visits.

In PHP, you can create, retrieve, and delete cookies using specific functions. Here's a breakdown of how cookies work in PHP:

1. Creating a Cookie

You create a cookie in PHP using the `setcookie()` function. Here's the basic syntax:

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

- **name**: The name of the cookie.
- **value**: The value of the cookie (stored as a string).
- **expire**: The time the cookie expires (Unix timestamp, e.g., `time() + 3600` for 1 hour).
- **path**: The path on the server where the cookie is available (e.g., `/` to make it available throughout the website).
- **domain**: The domain that the cookie is available to (e.g., `.example.com` to make it available across subdomains).
- **secure**: If `true`, the cookie will only be sent over HTTPS.
- **httponly**: If `true`, the cookie can only be accessed via HTTP and not by JavaScript (helps prevent XSS attacks).

Example of creating a cookie:

```
// Set a cookie that expires in 1 hour
setcookie("username", "JohnDoe", time() + 3600, "/");
```

2. Retrieving a Cookie

You can retrieve a cookie by accessing the `$_COOKIE` superglobal array. For example:

```
if (isset($_COOKIE['username'])) {
    echo "User is " . $_COOKIE['username'];
} else {
    echo "User not recognized!";
}
```

3. Deleting a Cookie

To delete a cookie, you set its expiration time to a past time. This will instruct the browser to delete the cookie.

```
// Delete the cookie by setting expiration time in the past
setcookie("username", "", time() - 3600, "/");
```

4. Key Points About Cookies

- Cookies are sent by the server to the browser with the `Set-Cookie` HTTP header.
- Cookies are stored in the user's browser and are sent back to the server with subsequent HTTP requests.
- They have size limitations (around 4 KB) and should not be used for storing sensitive information.
- They are ideal for maintaining session information or small amounts of data across pages or visits.

Session Handling in php

A **session** in PHP is a way to store information (in variables) to be used across multiple pages. Unlike cookies, which are stored on the user's computer, session data is stored on the server, and only a unique session ID is shared with the user's browser. This makes sessions more secure, as sensitive data does not need to be stored on the user's machine.

How PHP Sessions Work

1. **Session Initialization**: When a session starts, PHP generates a unique session ID and stores it in a cookie on the user's browser.
2. **Server-Side Storage**: All session data (e.g., user information, preferences, etc.) is stored on the server.
3. **Access Across Pages**: The session ID is passed between pages, allowing the user’s data to be accessed across different requests.

1. Starting a Session

Before using session variables, you must start a session using the `session_start()` function. This function must be called at the beginning of the script, before any HTML output.

```
<?php
// Start the session
session_start();
?>
```

2. Storing Data in Session Variables

Session variables are stored in the `$_SESSION` superglobal array. You can add data to this array to store it for the duration of the session.

Example:

```
<?php
session_start();

// Store session data
$_SESSION["username"] = "JohnDoe";
$_SESSION["email"] = "john@example.com";
?>
```

3. Accessing Session Variables

Once the session has started, you can access the session data stored in `$_SESSION` from any page:

```
<?php
session_start();

// Access session data
echo "Username: " . $_SESSION["username"];
echo "Email: " . $_SESSION["email"];
?>
```

4. Destroying a Session

To end a session and clear all session data, you use the `session_destroy()` function. However, before destroying a session, you must first start it with `session_start()`.

Example:

```
<?php
session_start();

// Unset all session variables
session_unset();

// Destroy the session
session_destroy();
?>
```

5. Key Points About Sessions

- **Security**: Sessions are more secure than cookies because session data is stored on the server, and only a session ID is passed between the browser and server.
- **Lifetime**: Sessions are temporary and are destroyed when the user closes their browser, unless configured otherwise. The session can also expire after a specific time of inactivity.
- **Server-Side Storage**: Session data is stored in temporary files on the server. This means you can store larger amounts of data compared to cookies.
- **Use Case**: Sessions are commonly used to store user authentication data (e.g., whether a user is logged in), user preferences, and shopping cart information.

Example of a Login System Using Sessions

```php
<?php
session_start();

// Simple login simulation
if ($_POST["username"] == "admin" && $_POST["password"] == "password") {
    // Store session data
    $_SESSION["loggedin"] = true;
    $_SESSION["username"] = $_POST["username"];
    echo "Welcome, " . $_SESSION["username"];
} else {
    echo "Invalid login!";
}
?>
```

In this example, when the user logs in, their login state and username are stored in the session, allowing them to access restricted pages across the site until the session is destroyed.

Unit III :Working With Database

Introduction to MySQL

MySQL is a relational database management system (RDBMS) that uses structured query language (SQL) to manage and manipulate data stored in databases. Here are some key features and concepts associated with MySQL in the context of databases:

Features of MySQL:

1. **Relational Database**: MySQL organizes data into tables that can be linked based on relationships, allowing for efficient data management.
2. **SQL Support**: MySQL uses SQL for querying and managing the data, providing a standard way to perform operations like selecting, inserting, updating, and deleting records.
3. **Open Source**: MySQL is open-source, meaning its source code is freely available, allowing for customization and community contributions.
4. **Scalability**: MySQL can handle large databases and high traffic, making it suitable for both small applications and large-scale enterprise solutions.
5. **Cross-Platform**: It runs on various operating systems, including Linux, Windows, and macOS.
6. **Transactions**: MySQL supports transactions, which ensure data integrity by allowing multiple operations to be executed as a single unit.
7. **Replication and Clustering**: MySQL offers features for data replication and clustering, enhancing availability and load balancing.

Basic Concepts:

- **Tables**: Data is organized into tables, which consist of rows (records) and columns (fields).
- **Primary Key**: A unique identifier for each record in a table.
- **Foreign Key**: A field that creates a link between two tables, establishing a relationship.
- **Indexes**: Structures that improve the speed of data retrieval operations on a database table.
- **Views**: Virtual tables created by querying data from one or more tables.

Use Cases:

MySQL is widely used in various applications, including:

- **Web Applications**: It powers many websites and web apps, especially those built on platforms like WordPress and Drupal.
- **Data Warehousing**: Used for analytical purposes and reporting.
- **E-commerce**: Manages product catalogs, customer data, and transactions.

.

Merits and Demerits MySQL

Merits MySQL

- MySQL is a Relational Database Management System or RDBMS which means that it stores and presents data in tabular form, organized in rows and columns.
- MySQL is more secure as it consists of a solid data security layer to protect sensitive data from intruders and passwords in MySQL are encrypted.
- MySQL is available for free to download and use from the official site of MySQL.
- MySQL is compatible with most operating systems, including Windows, Linux, NetWare, Novell, Solaris, and other variations of UNIX.

- MySQL provides the facility to run the clients and the server on the same computer or different computers, via the internet or local network.
- MySQL has a unique storage engine architecture which makes it faster, cheaper, and more reliable.
- MySQL gives developers higher productivity by using views, Triggers, and Stored procedures
- MySQL is simple and easy to use. You can build and interact with MySQL with only a basic knowledge of MySQL and a few simple SQL statements.
- MySQL has a client-server architecture. There can be any number of clients or application programs that communicate with the database server (MySQL) to query data, save changes, etc.
- MySQL is scalable and capable of handling more than 50 million rows. This is enough to handle almost any amount of data. Although the default file size limit is 4GB it can be increased to 8TB.
- MySQL allows transactions to be rolled back.
- MySQL is very flexible as it supports a large number of embedded applications.

Demerits MySQL

- MySQL is not very efficient in handling very large databases.
- MySQL doesn’t have as good a developing and debugging tool as compared to paid databases.
- MySQL versions less than 5.0 do not support COMMIT, stored procedure, and ROLE.
- MySQL is prone to data corruption as it is inefficient in handling transactions.
- MySQL does not support SQL check constraints.

In MySQL, data types define the kind of data that can be stored in a column, while constraints are rules that limit the types of data that can go into a table. Here's an overview of the common data types and constraints in MySQL:

Data Types

1. **Numeric Data Types**:
   - **INT**: Integer values. Variants include TINYINT, SMALLINT, MEDIUMINT, BIGINT.
   - **FLOAT**: Floating-point numbers.
   - **DOUBLE**: Double-precision floating-point numbers.
   - **DECIMAL**: Fixed-point numbers with a specified precision (e.g., DECIMAL(10,2) for 10 digits total, 2 of which are after the decimal point).
2. **String Data Types**:
   - **CHAR**: Fixed-length strings (e.g., CHAR(10) always stores 10 characters).
   - **VARCHAR**: Variable-length strings (e.g., VARCHAR(255) can store up to 255 characters).
   - **TEXT**: Large variable-length strings (up to 65,535 characters).
   - **BLOB**: Binary Large Object for storing binary data (e.g., images).
3. **Date and Time Data Types**:
   - **DATE**: Stores date values (YYYY-MM-DD).
   - **TIME**: Stores time values (HH:MM).
   - **DATETIME**: Combines date and time (YYYY-MM-DD HH:MM).

- **TIMESTAMP**: Stores a timestamp value (automatically updates with changes).
- **YEAR**: Stores year values (in YYYY format).

4. **Spatial Data Types**: Used for geometric data (e.g., POINT, LINESTRING, POLYGON).

Constraints

1. **NOT NULL**: Ensures that a column cannot contain NULL values.
2. **UNIQUE**: Ensures that all values in a column are different from one another.
3. **PRIMARY KEY**: A combination of NOT NULL and UNIQUE. It uniquely identifies each row in a table.
4. **FOREIGN KEY**: Creates a relationship between two tables, enforcing referential integrity by ensuring that a value in one table matches a value in another table.
5. **CHECK**: Ensures that all values in a column satisfy a specific condition (though support may vary by MySQL version).
6. **DEFAULT**: Sets a default value for a column if no value is provided during an insert operation.

Example Table Creation

Here's a simple example of creating a table with various data types and constraints:

Sql code:

```
CREATE TABLE Users (
   UserID INT AUTO_INCREMENT PRIMARY KEY,
   Username VARCHAR(50) NOT NULL UNIQUE,
   Password VARCHAR(255) NOT NULL,
   Email VARCHAR(100) NOT NULL UNIQUE,
   CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
   Age INT CHECK (Age >= 0),
   Country CHAR(2)
);
```

Basic SQL Commands

These basic MySQL commands will help you create databases and tables, manipulate data, and run queries effectively. Mastering these commands is essential for working with

Database Commands

1. **Create Database**: Create a new database.
   ```
   CREATE DATABASE database_name;
   ```
2. **Use Database**: Select a database to work with.
   ```
   USE database_name;
   ```
3. **Show Databases**: List all databases on the MySQL server.
   ```
   SHOW DATABASES;
   ```
4. **Drop Database**: Delete a database and all its contents.
   ```
   DROP DATABASE database_name;
   ```

Table Commands

1. **Create Table**: Create a new table.
   ```
   CREATE TABLE table_name (
   ```

```
    column1 datatype,
    column2 datatype,
    ...
);
```

2. **Show Tables**: List all tables in the selected database.
   SHOW TABLES;
3. **Describe Table**: Get information about the structure of a table.
   DESCRIBE table_name;
4. **Alter Table**: Modify an existing table (e.g., add or drop columns).
   ALTER TABLE table_name ADD column_name datatype;
   ALTER TABLE table_name DROP COLUMN column_name;
5. **Drop Table**: Delete a table and all its data.
   DROP TABLE table_name;

Data Manipulation Commands

1. **Insert Data**: Add new rows to a table.
   INSERT INTO table_name (column1, column2) VALUES (value1, value2);
2. **Select Data**: Retrieve data from a table.
   SELECT column1, column2 FROM table_name;
   SELECT * FROM table_name; -- To select all columns
3. **Update Data**: Modify existing rows.
   UPDATE table_name SET column1 = value1 WHERE condition;
4. **Delete Data**: Remove rows from a table.
   DELETE FROM table_name WHERE condition;

Querying Data

1. **Where Clause**: Filter records based on conditions.
   SELECT * FROM table_name WHERE condition;
2. **Order By**: Sort the result set.
   SELECT * FROM table_name ORDER BY column_name ASC; -- ASC for ascending, DESC for descending
3. **Limit**: Limit the number of records returned.
   SELECT * FROM table_name LIMIT number;
4. **Group By**: Group rows that have the same values in specified columns.
   SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;
5. **Having**: Filter groups based on a condition.
   SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name HAVING COUNT(*) > value;

Joins

1. **Inner Join**: Select records that have matching values in both tables.
   SELECT a.column1, b.column2
   FROM table_a a
   INNER JOIN table_b b ON a.common_column = b.common_column;
2. **Left Join**: Select all records from the left table and matched records from the right table.
   SELECT a.column1, b.column2
   FROM table_a a

```
LEFT JOIN table_b b ON a.common_column = b.common_column;
```

3. **Right Join**: Select all records from the right table and matched records from the left table.
```
SELECT a.column1, b.column2
FROM table_a a
RIGHT JOIN table_b b ON a.common_column = b.common_column;
```
4. **Full Outer Join**: Select all records when there is a match in either table.
```
SELECT a.column1, b.column2
FROM table_a a
FULL OUTER JOIN table_b b ON a.common_column = b.common_column;
```

MySQL Functions
These MySQLi functions are essential for interacting with a MySQL database in PHP. They allow you to establish a connection, execute queries, fetch results, and manage your database efficiently.

1. mysqli_connect()
This function is used to establish a connection to a MySQL database.
Syntax:
```
mysqli_connect(host, username, password, database, port, socket);
```
Example:
```
$conn = mysqli_connect("localhost", "your_username", "your_password",
"your_database");

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

2. mysqli_select_db()
This function selects a specific database to work with.
Syntax:
```
mysqli_select_db(connection, database);
```
Example:
```
if (!mysqli_select_db($conn, "your_database")) {
    die("Database selection failed: " . mysqli_error($conn));
}
```
Note: If you specify the database in mysqli_connect(), you don't need to use mysqli_select_db().

3. mysqli_query()
This function executes a SQL query against the database.
Syntax:
```
mysqli_query(connection, query);
```
Example:
```
$sql = "SELECT * FROM users";
$result = mysqli_query($conn, $sql);
```

```
if (!$result) {
    die("Query failed: " . mysqli_error($conn));
}
```

4. mysqli_num_rows()
This function returns the number of rows in a result set.
Syntax:
```
mysqli_num_rows(result);
```
Example:
```
$row_count = mysqli_num_rows($result);
echo "Number of users: " . $row_count;
```
5. mysqli_fetch_array()
This function fetches a result row as an associative array, a numeric array, or both.
Syntax:
```
mysqli_fetch_array(result, result_type);
```
Example:
```
while ($row = mysqli_fetch_array($result, MYSQLI_ASSOC)) {
    echo "Name: " . $row['name'] . " - Email: " . $row['email'] . "<br>";
}
```

6. mysqli_fetch_field()
This function returns information about a single field in a result set.
Syntax:
```
mysqli_fetch_field(result);
```
Example:
```
$fields = mysqli_query($conn, "SELECT * FROM users");
while ($field = mysqli_fetch_field($fields)) {
    echo "Field name: " . $field->name . "<br>";
}
```

7. mysqli_close()
This function closes the connection to the MySQL database.
Syntax:
```
mysqli_close(connection);
```

Example:
```
mysqli_close($conn);
```

Full Example
Here’s a complete example that demonstrates the use of these functions in a simple PHP script:
```
<?php
// 1. Connect to the database
$conn = mysqli_connect("localhost", "your_username", "your_password",
"your_database");
```

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

// 2. (Optional) Select the database if not specified in connect
// mysqli_select_db($conn, "your_database");

// 3. Execute a query
$sql = "SELECT * FROM users";
$result = mysqli_query($conn, $sql);

if (!$result) {
    die("Query failed: " . mysqli_error($conn));
}

// 4. Get number of rows
$row_count = mysqli_num_rows($result);
echo "Number of users: " . $row_count . "<br>";

// 5. Fetch and display results
while ($row = mysqli_fetch_array($result, MYSQLI_ASSOC)) {
    echo "Name: " . $row['name'] . " - Email: " . $row['email'] . "<br>";
}

// 6. Fetch field information
$fields = mysqli_query($conn, "SELECT * FROM users");
while ($field = mysqli_fetch_field($fields)) {
    echo "Field name: " . $field->name . "<br>";
}

// 7. Close the connection
mysqli_close($conn);
?>
```

Sub: Advanced PHP
UNIT _IV File Handling
What is File Handling in PHP?
File handling is the process of interacting with files on the server, such as reading file, writing to a file, creating new files, or deleting existing ones. File handling is essential for applications that require the storage and retrieval of data, such as logging systems, user-generated content, or file uploads.

Require () and include ()
PHP Include Files

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

PHP include and require Statements
It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

The include and require statements are identical, except upon failure:

require will produce a fatal error (E_COMPILE_ERROR) and stop the script
include will only produce a warning (E_WARNING) and the script will continue
So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of FrameWork, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax
```
include 'filename';

or

require 'filename';
```
PHP include Examples

Example 1
Assume we have a standard footer file called "footer.php", that looks like this:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
?>
```

To include the footer file in a page, use the include statement:

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>

</body>
</html>
```

Example 2
Assume we have a standard menu file called "menu.php":

```
<?php
echo '<a href="/default.asp">Home</a> -
<a href="/html/default.asp">HTML Tutorial</a> -
<a href="/css/default.asp">CSS Tutorial</a> -
<a href="/js/default.asp">JavaScript Tutorial</a> -
<a href="default.asp">PHP Tutorial</a>';
?>
```

All pages in the Web site should use this menu file. Here is how it can be done (we are using a <div> element so that the menu easily can be styled with CSS later):

Example

```
<html>
<body>

<div class="menu">
<?php include 'menu.php';?>
</div>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
```

```
</body>
</html>
```
Example 3
Assume we have a file called "vars.php", with some variables defined:

```
<?php
$color='red';
$car='BMW';
?>
```
Then, if we include the "vars.php" file, the variables can be used in the calling file:

Example
```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php include 'vars.php';
echo "I have a $color $car.";
?>

</body>
</html>
```
PHP include vs. require
The require statement is also used to include a file into the PHP code.

However, there is one big difference between include and require; when a file is included with the include statement and PHP cannot find it, the script will continue to execute:

Example
```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php include 'noFileExists.php';
echo "I have a $color $car.";
?>

</body>
</html>
```
If we do the same example using the require statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error:

Example

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php require 'noFileExists.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

PHP File Permissions

Summary: in this tutorial, you will learn how to deal with PHP file permissions, including checking and changing file permissions.

File permissions specify what a user can do with a file, e.g., reading, writing, or executing it. Notice that PHP automatically grants appropriate permissions behind the scenes.

For example, if you create a new file for writing, PHP automatically grants the read and write permissions.

PHP provides some useful functions for checking and changing the file permissions.

Checking file permissions

PHP has three handy functions that check file permissions:

is_readable() function returns true if the file exists and is readable; otherwise, it returns false.
is_writable() function returns true if the file exists and is writable; otherwise, it returns false.
is_executable() function returns true if the file exists and executable; otherwise, it returns false.
Let’s take a look at the following example:

```
<?php

$filename = 'readme.txt';

$functions = [
        'is_readable',
        'is_writable',
        'is_executable'
];

foreach ($functions as $f) {
	echo "$f($filename) .The file  . $filename . $f :";
}
```

Output:

```
is_readable(readme.txt) .The file . readme.txt . is_readable
is_writable(readme.txt) .The file . readme.txt . is_writable
is_executable(readme.txt) .The file . readme.txt . is_executable
```

Besides those functions, PHP also provides the fileperms() function that returns an integer, which represents the permissions set on a particular file. For example:

```
<?php

$permissions = fileperms('readme.txt');

echo substr(sprintf('%o', $permissions), -4); //0666
```

Changing file permissions

To change the file permission or mode, you use the chmod() function:

```
chmod ( string $filename , int $permissions ) : bool
```

Code language: PHP (php)

The chmod() function has two parameters:

$filename is the file that you want to change the permissions.
$permissions parameter consists of three octal number components that specify access restrictions for the owner, the user group in which the owner is in, and everyone else in this sequence.
The chmod() function returns true on success or false on failure.

The permissions argument is represented by an octal number that contains three digits:

The first digit specifies what the owner of the file can read, write, or execute the file.
The second digit specifies what the user group in which the owner is in can read, write, or execute the file.
The third digit specifies what everyone else can read, write, or execute the file.
The following table illustrates the value of each digit that represents the access permission for particular users (owner, user group, or everyone else) :

Value	**Permission**
0	cannot read, write or execute
1	can only execute
2	can only write
3	can write and execute
4	can only read
5	can read and execute
6	can read and write
7	can read, write and execute

The following example sets permission that the only owner can read and write a file, everyone else only can read the file:

```
<?php
$filename = './readme.txt';
chmod($filename, 0644);
```

Notice that we put 0 before 644 to instruct PHP to treat it as an octal number.

Summary

Use the is_readable(), is_writable(), is_executable() to check if a file exists and readable, writable, and executable.
Use the chmod() function to set permissions for a file.

Common File Handling Functions in PHP

What is File Handling in PHP?

File handling in PHP is used to you to create, open, read, write, delete, and manipulate files on a server. It is used when you need to store data persistently or handle files uploaded by users. PHP provides several built-in functions to make file handling easy and secure.

Opening and Closing Files

Reading from Files
Writing to Files
Deleting Files
Common File Handling Functions in PHP
fopen() – Opens a file
fclose() – Closes a file
fread() – Reads data from a file
fwrite() – Writes data to a file
file_exists() – Checks if a file exists
unlink() – Deletes a file

Opening and Closing Files

Before you can read or write to a file, you need to open it using the fopen() function, which returns a file pointer resource. Once you're done working with the file, you should close it using fclose() to free up resources.

```
<?php

// Open the file in read mode
$file = fopen("gfg.txt", "r");

if ($file) {
   echo "File opened successfully!";
   fclose($file); // Close the file
} else {
   echo "Failed to open the file.";
}
?>
```

File Modes in PHP

Files can be opened in any of the following modes:

“w” – Opens a file for writing only. If file does not exist then new file is created and if file already exists then file will be truncated (contents of file is erased).
“r” – File is open for reading only.
“a” – File is open for writing only. File pointer points to end of file. Existing data in file is preserved.
“w+” – Opens file for reading and writing both. If file not exist then new file is created and if file already exists then contents of file is erased.
“r+” – File is open for reading and writing both.
“a+” – File is open for write/read. File pointer points to end of file. Existing data in file is preserved. If file is not there then new file is created.
“x” – New file is created for write only.
Reading from Files
There are two ways to read the contents of a file in PHP. These are –

1. Reading the Entire File

You can read the entire content of a file using the fread() function or the file_get_contents() function.

```
<?php

$file = fopen("gfg.txt", "r");
$content = fread($file, filesize("gfg.txt"));

echo $content;
fclose($file);
?>
```

2. Reading a File Line by Line

You can use the fgets() function to read a file line by line.

```
<?php

$file = fopen("gfg.txt", "r");

if ($file) {
    while (($line = fgets($file)) !== false) {
        echo $line . "<br>";
    }
    fclose($file);
}

?>
```

3. Writing Files

You can write to files using the fwrite() function. It writes data to an open file in the specified mode.

```
<?php

// Open the file in write mode
$file = fopen("gfg.txt", 'w');

if ($file) {
    $text = "Hello world\n";
    fwrite($file, $text);
      fclose($file);
}

?>
```

4.Deleting Files

Use the unlink() function to delete the file in PHP.

```
<?php
if (file_exists("gfg.txt")) {
    unlink("gfg.txt");
    echo "File deleted successfully!";
} else {
echo "File does not exist.";
}
?>
```