

ARBA MINCH UNIVERSITY



INSTITUTE OF TECHNOLOGY

FACULTY OF COMPUTING AND SOFTWARE ENGINEERING

Internet Programming II Lecture Material

By: Chala Simon

ARBA MINCH UNIVERSITY
INSTITUTE OF TECHNOLOGY

FACULTY OF COMPUTING AND SOFTWARE ENGINEERING

Course Title Internet Programming II

Course Code ITec3093

CP 5 (2hr Lecture, 3hr Laboratory)

Module Title: Web Systems and Technologies

Learning Outcomes:

At the end of this course the students will be able to:

- Understand server-side scripting
- Develop web-based applications
- Create Forms on Websites
- Connect Webpages to databases
- Design web page for e-commerce

Table of Contents

Introduction to server-side programming.....	1
What is server-side programming?.....	1
Advantages of Server-Side Scripting over Client-Side Scripting:	1
Server-Side Scripting Languages	2
Introduction to PHP	3
PHP: PHP Basic syntax	4
PHP Syntax.....	4
Writing PHP Code in an HTML Document	4
Declaring Variables in PHP	4
Send Data to the Web Browser.....	5
Outputting Text and HTML with PHP	5
Generating Dynamic Content with PHP.....	5
Writing Comments in PHP	5
Utilizing Variables in PHP	6
Concatenating Variables and Strings in PHP	6
Manipulating Numbers and Working with Constants in PHP	6
Manipulating Numbers in PHP.....	6
Working with Constants in PHP:	7
Manipulating Arrays in PHP	7
Arrays in PHP.....	7
Creating Arrays in PHP	7
Accessing Array Elements in PHP	7
Adding Elements to an Array in PHP	8
Removing Elements from an Array in PHP.....	8
String Manipulation in PHP.....	8
Strings in PHP	8
Creating Strings in PHP	8
Concatenating Strings in PHP.....	8
String Functions in PHP	9
Working with Functions.....	9
Functions in PHP.....	9
Creating Functions in PHP.....	9
Calling Functions in PHP	10
Passing Arguments to Functions:.....	10
Returning Values from Functions:	10

HTML Forms and Server-Side Scripting	11
Overview of HTML Forms.....	11
How to create HTML forms.....	11
Processing Form Data	13
Use Conditionals and Operators.....	14
PHP Form Validation	17
Send Values to a Script Manually	20
Work with Forms and arrays of data	21
Loops in PHP	22
Files and Directories	24
PHP include and require Statements.....	24
Testing for the Existence of Files	25
Opening the file	26
Read from Files.....	27
Write to Files.....	30
Working with directories.....	31
Upload Files	33
Manipulating Databases with PHP.....	36
Connecting to a MySQL Database.....	36
Connecting to a MySQL Database:.....	38
Sending Data to a MySQL Database:.....	39
Retrieving Data from a MySQL Database	40
Modifying and Removing Data from a MySQL Database:.....	41
Introduction to PHP Sessions and Cookies.....	42
Understanding statelessness and the need for session management.....	42
PHP Session	42
PHP Cookies.....	45
References	47
Model Questions & Answers	48
Questions.....	48
Answer keys	51

Introduction to server-side programming

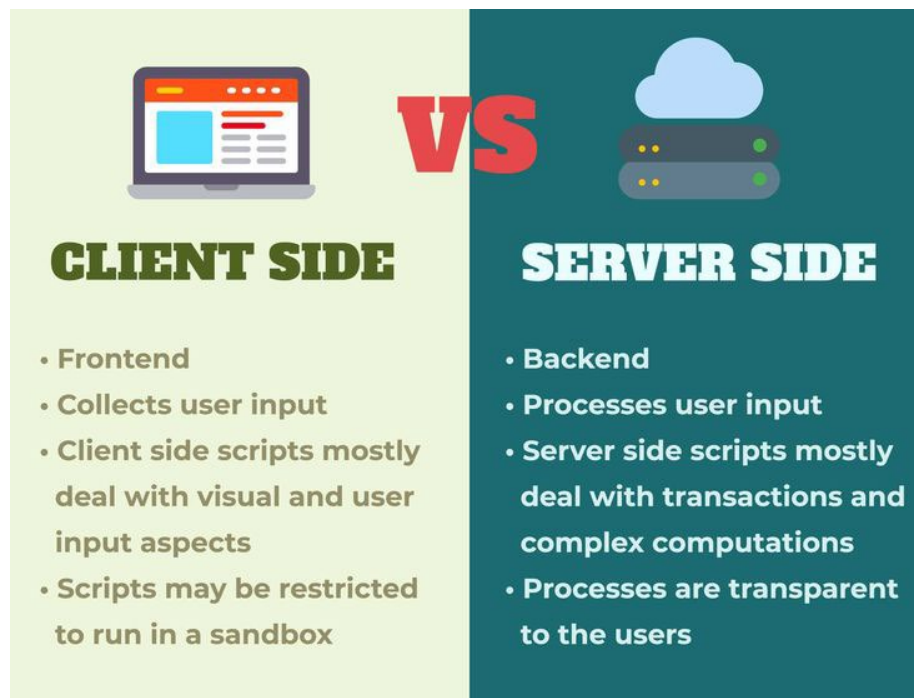
What is server-side programming?

Server-side programming refers to the execution of code on a web server to generate dynamic web pages.

It involves processing user requests, interacting with databases, and generating output that is sent back to the client's web browser.

Server-side programming is used to create interactive web applications that require dynamic content.

Unlike client-side scripting, which runs on the client's machine, server-side scripting allows for more advanced functionality and data processing.



Advantages of Server-Side Scripting over Client-Side Scripting:

- **Enhanced Functionality:** Server-side scripting provides advanced functionality and processing capabilities that are not possible with client-side scripting alone. It enables server-side operations such as database queries, data processing, and complex business logic.
- **Database Connectivity:** Server-side scripting languages have built-in features for connecting to databases, retrieving data, and manipulating it as needed. This makes it easier to work with data-driven web applications and perform tasks like user authentication, content management, and e-commerce transactions.

- **Security:** Server-side scripting allows for secure processing of sensitive information and implementation of access control mechanisms. By keeping critical operations on the server, it helps protect the integrity of data and ensures secure transactions.
- **Code Reusability:** Server-side scripts can be organized into functions and modules, promoting code reusability and modularity. This saves development time and effort as developers can leverage existing code for similar tasks across different web pages or applications.

Examples of Dynamic Web Applications and Their Functionality:

- **E-commerce Websites:** Dynamic e-commerce websites use server-side scripting to handle product catalogs, user authentication, shopping cart functionality, order processing, and payment integration.
- **Content Management Systems (CMS):** CMS platforms use server-side scripting to manage and display dynamic content, handle user permissions, and provide a flexible and customizable interface for website administration.
- **Social Networking Platforms:** Social media websites rely on server-side scripting to process user-generated content, handle connections between users, implement real-time updates, and manage privacy settings.
- **Online Banking Systems:** Server-side scripting enables secure authentication, transaction processing, account management, and data encryption in online banking applications.
- **Web-based Forms and Surveys:** Server-side scripting is used to validate form input, process submitted data, and store it in databases. It allows for data verification, error handling, and data analysis for survey and feedback applications.

Server-Side Scripting Languages

There are several popular server-side scripting languages used in web development. Here is an overview of some of the widely used languages: PHP, Python, Ruby, and ASP.NET.

1. **PHP (Hypertext Preprocessor):** - PHP is one of the most widely used server-side scripting languages, especially for web development.
2. **Python:** - Python is a versatile language used in various domains, including web development.
3. **Ruby:** - Ruby is a dynamic, object-oriented scripting language known for its simplicity and productivity.
4. **ASP.NET:** - ASP.NET is a server-side web application framework developed by Microsoft.

Factors to Consider when Choosing a Server-Side Scripting Language:

- **Project Requirements:** Consider the specific requirements of your project, such as performance, scalability, database integration, or third-party library support.
- **Learning Curve:** Evaluate the learning curve associated with each language and whether your team has prior experience or familiarity with any particular language.
- **Community and Support:** Consider the size and activity of the language's community, availability of documentation, forums, and online resources.
- **Ecosystem and Frameworks:** Assess the availability and maturity of frameworks and libraries that align with your project's needs.
- **Integration:** Consider the compatibility of the language with other systems or technologies you plan to use, such as databases, server environments, or APIs.

It's important to choose a server-side scripting language that aligns with your project's requirements, team expertise, and long-term scalability. Evaluating the features, syntax, community support, and considering the factors mentioned above will help you make an informed decision.

Introduction to PHP

PHP (Hypertext Preprocessor) is a popular server-side scripting language used to create dynamic web pages. It was originally created in 1994 by **Rasmus Lerdorf** and has since become one of the most widely used programming languages on the web.

PHP is an open-source language and is constantly evolving, with new features and improvements being added regularly. Some of the key features of PHP include:

- Compatibility with various web servers and operating systems
- Support for a wide range of databases
- Easy integration with HTML, CSS, and JavaScript
- Large community of developers
- Components Required to Set up a PHP Development Environment:

To develop PHP applications, you will need the following components:

- **Web Server:** A web server is software that allows you to serve web pages to clients over the internet. Some of the popular web servers include Apache, Nginx, and IIS.
- **PHP Interpreter:** The PHP interpreter is the software that reads and executes PHP code. You can download the latest version of the PHP interpreter from the official PHP website.
- **Database Management System:** PHP is commonly used to interact with databases, so you will need a database management system such as MySQL or PostgreSQL.
- **Text Editor or Integrated Development Environment (IDE):** A text editor or IDE is used to write and edit PHP code. Some popular options include *Visual Studio Code*, *Sublime Text*, and *PhpStorm*.

PHP: PHP Basic syntax

PHP Syntax

PHP code is executed on the server-side and can be embedded within an HTML document. PHP code is enclosed in PHP tags, which are typically "<?php" and "?>". The code within the tags is executed by the server and the output is sent to the web browser.

Writing PHP Code in an HTML Document

To write PHP code in an HTML document, you must enclose the PHP code within the PHP tags. For example, to display the current date in a web page, you can use the following PHP code:

```
<html>
<head>
    <title>PHP Date Example</title>
</head>
<body>
    <h1>Today's Date</h1>
    <?php
        echo date("Y/m/d");
    ?>
</body>
</html>
```

In the above example, the PHP code to display the current date is embedded within the HTML tags. The "echo" statement is used to output the result to the web page.

Declaring Variables in PHP

Variables are used to store data in PHP. In PHP, variables are declared using the "\$" symbol followed by the variable name. For example, to declare a variable called "name", you would use the following syntax:

```
$name = "John";
```

In the above example, the variable "name" is assigned the value "John". You can also declare variables with a specific data type, such as integer or float. For example:

```
$age = 30;
$price = 9.99;
```

PHP is a loosely typed language, which means that you do not need to declare the data type of a variable before using it. The data type is automatically determined based on the value assigned to the variable.

Send Data to the Web Browser

Outputting Text and HTML with PHP

PHP can be used to output text and HTML to the web browser. The "echo" statement is used to output text or HTML to the web page. For example, to output the text "Hello, World!" to the web page, you can use the following PHP code:

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

In the above example, the "echo" statement outputs the text "Hello, World!" to the web page.

You can also use PHP to output HTML to the web page. For example, to output a heading and a paragraph of text, you can use the following PHP code:

```
<?php
    echo "<h1>Welcome to My Website</h1>";
    echo "<p>Thank you for visiting! </p>";
?>
```

In the above example, the "echo" statements output HTML code to the web page, which is then rendered by the web browser.

Generating Dynamic Content with PHP

PHP can be used to generate dynamic content based on user input or other variables. For example, you can use PHP to generate a personalized greeting based on the user's name. To do this, you would first need to capture the user's name using a form or other input method. Once you have the user's name, you can use PHP to generate the personalized greeting. For example:

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

In the above example, the \$_POST superglobal variable is used to capture the user's name from a form. The variable \$name is then used to generate the personalized greeting using the "echo" statement.

Writing Comments in PHP

Comments are used to explain what a particular section of code does or to provide additional information about the code. In PHP, comments can be written using two different methods. Single-line comments can be written using two forward slashes ("//"), while multi-line comments can be written using /* and */.

```
// This is a single-line comment
```

```
/*  
This is a  
multi-line comment  
*/
```

Comments should be used to make code more readable and easier to understand, both for yourself and for other developers who may be working with your code in the future.

Utilizing Variables in PHP

Variables are used to store data in PHP. You can use variables to store strings, numbers, and other data types. To declare a variable, you use the "\$" symbol followed by the variable name. For example:

```
$name = "John";  
  
$age = 30;
```

You can then use the variables in your code, either by referencing the variable name directly or by concatenating the variable with other strings or variables. For example:

```
echo "My name is " . $name . " and I am " . $age . " years old.";
```

In the above example, the variables \$name and \$age are concatenated with other strings to create a sentence that is output to the web page.

Concatenating Variables and Strings in PHP

To concatenate variables and strings in PHP, you use the "." operator. For example:

```
$name = "John";  
  
echo "Hello, " . $name . "!";
```

In the above example, the variable \$name is concatenated with the string "Hello, " to create a personalized greeting that is output to the web page.

Manipulating Numbers and Working with Constants in PHP

Manipulating Numbers in PHP

PHP provides a wide range of functions for manipulating numbers. You can perform mathematical operations such as addition, subtraction, multiplication, and division using basic arithmetic operators. For example:

```
$x = 10;  
$y = 5;  
$sum = $x + $y; // adds x and y together  
$difference = $x - $y; // subtracts y from x  
$product = $x * $y; // multiplies x and y together  
$quotient = $x / $y; // divides x by y
```

You can also use functions such as `round()`, `ceil()`, and `floor()` to round numbers to the nearest whole number, to the next highest whole number, or to the next lowest whole number, respectively.

Working with Constants in PHP:

Constants are values that cannot be changed once they are defined. In PHP, you can define constants using the `define()` function. For example:

```
define("PI", 3.14159);  
echo PI; // outputs 3.14159
```

In the above example, the constant "PI" is defined as 3.14159 using the `define()` function. Once a constant is defined, it can be used throughout your code without the risk of accidentally changing its value.

You can also define constants with arrays:

```
define("COLORS", ['red', 'green', 'blue']);  
echo COLORS[0]; // outputs "red"
```

In the above example, the constant "COLORS" is defined as an array of three colors. The first color in the array, "red," is output to the web page using the index notation.

Manipulating Arrays in PHP

Arrays in PHP

An array is a collection of variables that are stored together under a single name. In PHP, you can create arrays that store different types of data, including numbers, strings, and even other arrays.

Creating Arrays in PHP

You can create an array in PHP by using the `array()` function. The basic syntax of the `array()` function is as follows:

```
$array_name = array(value1, value2, value3, ...);
```

For example:

```
$fruits = array("apple", "banana", "orange");
```

In the above example, an array called `$fruits` is created, which contains the values "apple", "banana", and "orange".

Accessing Array Elements in PHP

You can access the elements of an array in PHP using their index values. The index of an array starts at 0, which means that the first element of an array has an index of 0, the second element has an index of 1, and so on.

For example:

```
$fruits = array("apple", "banana", "orange");
```

```
echo $fruits[0]; // outputs "apple"
echo $fruits[1]; // outputs "banana"
echo $fruits[2]; // outputs "orange"
```

Adding Elements to an Array in PHP

You can add new elements to an array in PHP using the `array_push()` function. The basic syntax of the `array_push()` function is as follows:

```
array_push($array_name, value1, value2, ...);
```

For example:

```
$fruits = array("apple", "banana", "orange");
array_push($fruits, "grape", "kiwi");
print_r($fruits); // outputs Array ( [0] => apple [1] => banana [2]
=> orange [3] => grape [4] => kiwi )
```

In the above example, the `array_push()` function adds the values "grape" and "kiwi" to the `$fruits` array.

Removing Elements from an Array in PHP

You can remove elements from an array in PHP using the `unset()` function. The basic syntax of the `unset()` function is as follows:

```
unset($array_name[index]);
```

For example:

```
$fruits = array("apple", "banana", "orange");
unset($fruits[1]);
print_r($fruits); // outputs Array ( [0] => apple [2] => orange )
```

In the above example, the `unset()` function removes the element at index 1 from the `$fruits` array, which is the value "banana".

String Manipulation in PHP

Strings in PHP

A string is a sequence of characters that are used to represent text. In PHP, you can create strings using single quotes (') or double quotes ("). Single quotes are used to create literal strings, while double quotes allow you to include variables and special characters in a string.

Creating Strings in PHP

You can create a string in PHP by enclosing it in either single or double quotes. For example:

```
$name = 'John';
$message = "Hello $name!";
```

In the above example, the variable `$name` contains the string 'John', and the variable `$message` contains the string "Hello John!".

Concatenating Strings in PHP

You can concatenate (join) two or more strings in PHP using the dot (.) operator. For example:

```
$first_name = 'John';  
$last_name = 'Doe';  
$name = $first_name . ' ' . $last_name;  
echo $name; // outputs "John Doe"
```

In the above example, the dot (.) operator is used to concatenate the `$first_name` and `$last_name` variables, separated by a space.

String Functions in PHP

PHP provides a number of built-in functions for manipulating strings. Here are some examples:

- **`strlen()`**: Returns the length of a string.

```
$str = 'Hello World!';  
echo strlen($str); // outputs 12
```

- **`strpos()`**: Returns the position of the first occurrence of a substring in a string.

```
$str = 'Hello World!';  
echo strpos($str, 'World'); // outputs 6
```

- **`str_replace()`**: Replaces all occurrences of a substring in a string with another substring.

```
$str = 'Hello World!';  
echo str_replace('World', 'Universe', $str); // outputs "Hello Universe!"
```

Working with Functions

Functions in PHP

A function is a block of code that performs a specific task. Functions are used to organize code and make it more reusable. In PHP, you can create your own functions or use built-in functions that are provided by the PHP language.

Creating Functions in PHP

You can create a function in PHP using the `function` keyword, followed by the name of the function and any arguments that it takes. For example:

```
function sayHello($name) {  
    echo "Hello, $name!";  
}
```

In the above example, the function `sayHello` takes one argument, `$name`, and echoes out a message that includes the value of the `$name` argument.

Calling Functions in PHP

You can call a function in PHP by using its name and passing any required arguments. For example:

```
$name = 'John';  
sayHello($name); // outputs "Hello, John!"
```

In the above example, the function sayHello is called with the argument \$name, which is set to the value 'John'.

Passing Arguments to Functions:

You can pass one or more arguments to a function in PHP. When you call a function, you can pass any required arguments inside parentheses, separated by commas. For example:

```
function multiply($a, $b) {  
    return $a * $b;  
}  
$result = multiply(3, 5); // returns 15
```

In the above example, the function multiply takes two arguments, \$a and \$b, and returns their product. The function is called with the arguments 3 and 5, and the result is stored in the variable \$result.

Returning Values from Functions:

A function in PHP can also return a value. To return a value from a function, use the return keyword followed by the value that you want to return. For example:

```
function add($a, $b) {  
    $sum = $a + $b;  
    return $sum;  
}  
$result = add(3, 5); // returns 8
```

In the above example, the function add takes two arguments, \$a and \$b, and returns their sum. The function is called with the arguments 3 and 5, and the result is stored in the variable \$result.

HTML Forms and Server-Side Scripting

Overview of HTML Forms

A form is a document with blank fields for a user to insert or update data. The user's data is stored in the database and retrieved anytime.

Using forms to collect data in a web application is a simple task.

Some of the popular forms include:

- Contact Forms
- Search Forms
- Login Forms
- Registration Forms

The form is presented to the user who inserts data and submits the form using a submit button. Most of the time, the form data is sent to the server for processing. Processing user input involves validating inputs, database operations, and providing feedback to the user. There are four database operations involved, these being create, read, update, and delete. This pattern is commonly known by the acronym CRUD operations.

Hypertext Transfer Protocol (HTTP) enables communication between the client (browser) and the server. An HTTP request is sent by a client to the server which then returns a response. Though HTTP supports several methods, we will focus on GET, POST, and PUT. Data is processed based on the selected method.

How to create HTML forms

HTML forms can contain special elements such as buttons, radio buttons, and checkboxes. It, therefore, becomes easy for the user to interact with the webpage. Forms should be user-friendly. This means that a user with no technical skills should be able to use it.

Forms are defined by the `<form></form>` tags. The form tag surrounds all the inputs. It also gives instructions about how and where to submit the form. The HTML form sends data to your PHP script using either POST or GET methods.

Here are the lists some popular HTML form element tags:

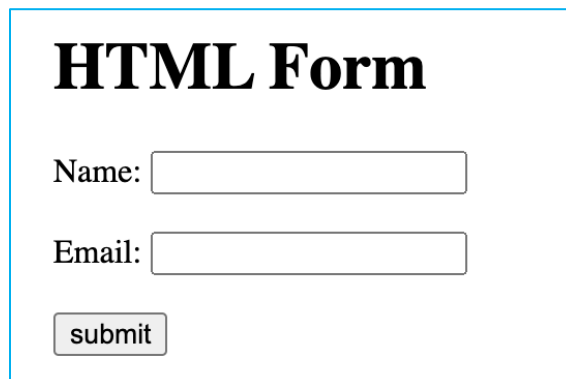
Tag	Description
<code><form></code>	It defines an HTML form to enter inputs by the used side.
<code><input></code>	It defines an input control.
<code><textarea></code>	It defines a multi-line input control.
<code><label></code>	It defines a label for an input element.

<fieldset>	It groups the related element in a form.
<legend>	It defines a caption for a <fieldset> element.
<select>	It defines a drop-down list.
<optgroup>	It defines a group of related options in a drop-down list.
<option>	It defines an option in a drop-down list.
<button>	It defines a clickable button.

Here is an example of a form that submits data to a file named index.php. To have a complete source code, use this HTML code and change the method to either **POST** or **GET** where necessary.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>HTML Form</title>
</head>
<body>
  <h1>HTML Form</h1>
  <form method="POST" action="process.php">
    Name: <input type="text" name="name"><br><br>
    Email: <input type="text" name="email"><br>
    <br>
    <input type="submit" value="submit" >
  </form>
</body>
</html>
```

The output for the above code is as shown in the screenshot below.



The screenshot displays the rendered HTML form. It features a main heading 'HTML Form'. Below the heading, there are two text input fields. The first is labeled 'Name:' and the second is labeled 'Email:'. At the bottom of the form, there is a button labeled 'submit'.

The **action** identifies the page where the form input is submitted. Data can be submitted on the same page as the form or on a different page. The method specifies how data is processed. This can be **POST**, **GET**, or **PUT**. The **GET** method collects data from the server and sends it in the URL. The data submitted via the **POST** method is stored in the HTTP request body and cannot be seen on the URL.

However, HTML does not natively support the **PUT** method for form submissions. The **PUT** method is typically used in RESTful APIs to update or replace existing resources.

If you need to use the **PUT** method in your form submission, you may consider using JavaScript or an **AJAX** request to perform the submission programmatically. By capturing the form data and constructing a custom HTTP request, you can send the data to the server-side script using the **PUT** method.

Processing Form Data

- Create a PHP script that will handle the form submission (`process.php` in the example).
- Access the form data using the `\$_POST` or `\$_GET` **superglobals** based on the form's **method** attribute.
- Validate and sanitize the form inputs to ensure data integrity and security.
- Perform any necessary actions or processing based on the form data.

Example (`process.php`):

```
<?php
// getting form data
$name = $_POST['name'];
$email = $_POST['email'];

// Validate and sanitize the form inputs
$name = trim($name);
$email = filter_var($email, FILTER_SANITIZE_EMAIL);

// Perform actions with the form data
// e.g., store in a database, send an email, etc.

// Display a success message
echo "Form submitted successfully!";
?>
```

- PHP **superglobal** arrays (global variables) contain information about the current request, server, etc.:

Array	Description
<code>\$_GET</code> , <code>\$_POST</code>	parameters passed to GET and POST requests

\$_REQUEST	parameters passed to any type of request
\$_SERVER, \$_ENV	information about the web server
\$_FILES	files uploaded with the web request
\$_SESSION, \$_COOKIE	"cookies" used to identify the user (seen later)

- These are special kinds of arrays called **associative arrays**.

Working with forms in PHP allows you to collect user input, validate and sanitize the data, and perform actions based on the submitted data. By following the outlined steps, you can create interactive and functional web forms.

Use Conditionals and Operators

Conditional statements allow you to execute code only if a certain condition is met.

In PHP, like in many other programming languages, you can use conditional statements to make decisions and execute different blocks of code based on certain conditions. These conditional statements allow you to control the flow of your program and perform specific actions based on the evaluation of logical or comparative test conditions.

1. The **if** Statement: The **if** statement is the most basic conditional statement in PHP. It allows you to execute a block of code only if a specified condition is true. If the condition is false, the code block is skipped.

Example:

```
$age = 25;

if ($age >= 18) {
    echo "You are an adult.";
}
```

In the above example, the code inside the **if** statement will be executed only if the variable **\$age** is greater than or equal to 18. If the condition is true, it will output "You are an adult."

2. The **if...else** Statement: The **if...else** statement extends the **if** statement by providing an alternative code block to execute when the condition is false.

Example:

```
$age = 15;
if ($age >= 18) {
    echo "You are an adult.";
} else {
    echo "You are not yet an adult.";
}
```

In the above example, if the condition **\$age >= 18** is false, the code inside the **else** block will be executed. It will output "You are not yet an adult."

3. The **if...elseif...else** Statement: The **if...elseif...else** statement allows you to test multiple conditions and execute different code blocks based on the evaluation of each condition. It provides a way to handle multiple possible scenarios.

Example:

```
$score = 85;
if ($score >= 90) {
    echo "You got an A.";
} elseif ($score >= 80) {
    echo "You got a B.";
} elseif ($score >= 70) {
    echo "You got a C.";
} else {
    echo "You need to improve your score.";
}
```

In the above example, the code checks the value of the **\$score** variable and executes the corresponding code block based on the condition that evaluates to true. If none of the conditions are true, the code inside the **else** block will be executed.

4. The **switch...case** Statement: The **switch...case** statement provides an alternative way to handle multiple conditions. It allows you to compare a single variable against multiple possible values and execute different code blocks based on the matching value.

Example:

```
$day = "Monday";
switch ($day) {
    case "Monday":
        echo "It's the start of the week.";
        break;
    case "Friday":
        echo "It's the end of the week.";
        break;
    default:
        echo "It's a regular day.";
        break;
}
```

In the above example, the code compares the value of the variable **\$day** against different cases. If a match is found, the corresponding code block is executed. If none of the cases match, the code inside the **default** block will be executed.

These conditional statements provide you with the flexibility to control the execution of your PHP code based on specific conditions and make your programs more dynamic and responsive to different scenarios.

PHP Comparison and logical operators

Symbol	Meaning	Type	Example
==	is equal to	comparison	\$x == \$y
!=	is not equal to	comparison	\$x != \$y
<	less than	comparison	\$x < \$y
>	greater than	comparison	\$x > \$y
<=	less than or equal to	comparison	\$x <= \$y
>=	greater than or equal to	comparison	\$x >= \$y
!	not	logical	!\$x
&&	and	logical	\$x && \$y
AND	and	logical	\$x and \$y
	or	logical	\$x \$y
OR	or	logical	\$x or \$y
XOR	and not	logical	\$x XOR \$y

PHP Form Validation

Form validation is essential to ensure the **integrity** and **accuracy** of data submitted through web forms. It helps **prevent errors**, **inconsistencies**, and **security vulnerabilities** in your application. Proper validation of form data is important to *protect your form from hackers and spammers*. It also *enhances the user experience* by providing feedback and guiding users to provide valid data.

PHP is a server-side scripting language specifically designed for web development, making it well-suited for form validation tasks. PHP offers a wide range of built-in functions and libraries that simplify the form validation process.

The validation process involves checking user-submitted data against predefined rules or criteria. PHP validates the form inputs on the server-side before processing or storing the data. It typically includes checking for required fields, validating data types and formats, enforcing length and size constraints, and implementing custom validation rules.

Generally, there are two goals in PHP form validation:

- The first goal of form validation is to check if something was entered or selected in form elements.
- The second goal is to ensure that the submitted data meets the following criteria:
 - It is of the **correct type** (numeric, string, etc.).

- It is in the **right format** (such as a valid email address).
- It matches **specific acceptable values** (for example, \$gender being equal to either "M" or "F").

Validation Technique	Description	Example
Required Fields	Check if fields are empty and display error messages if necessary.	<pre>if(empty(\$_POST['name'])) { echo "Name is required."; }</pre>
Data Types and Formats	Validate input based on specific data types or formats (e.g. email addresses, numeric input, URLs).	<pre>if(!filter_var(\$_POST['email'], FILTER_VALIDATE_EMAIL)) { echo "Invalid email format."; }</pre>
Length and Size Constraints	Set minimum and maximum lengths or limit file sizes.	<pre>if(strlen(\$_POST['password']) < 8) { echo "Password must be at least 8 characters."; }</pre>
Custom Validation Rules	Implement custom validation functions to validate input based on specific business rules.	<pre>function validateAge(\$age) { if(\$age < 18) { return "You must be at least 18 years old."; } }</pre>

isset() vs empty() function

The **isset()** and **empty()** functions are commonly used in PHP to check the status and value of variables. While they serve similar purposes, there are differences in their behavior and use cases.

isset() function:

- The **isset()** function checks if a variable is set and has a non-null value.
- It returns **true** if the variable exists and has a value, even if the value is 0, false, or an empty string.
- If the variable is not set or has a value of **null**, **isset()** will return **false**.

empty() function:

- The **empty()** function checks if a variable is considered to be empty.
- It returns **true** if the variable is empty. Variables are considered empty if they are not set, have a value of null, false, 0, or an empty string ("").
- If the variable is non-empty, **empty()** will return **false**.

Generally,

- **isset()** is used *to check if a variable is set and has a value*,
- while **empty()** is used to check if a variable is empty.

It's important to choose the appropriate function based on your specific validation requirements.

Introduction to regular expressions

Regular expressions (regex) in PHP are powerful tools for pattern matching and manipulating strings. They provide a concise and flexible way to validate and process text data based on specific patterns. Regular expressions are widely used for form validation, data extraction, search operations, and more.

Regular expressions consist of a combination of literal characters, metacharacters, and quantifiers. In PHP, regular expressions are defined as strings, enclosed between delimiters (often '/').

Basic Regex Functions in PHP:

- **preg_match():** Tests if a pattern matches a string.
- **preg_match_all():** Finds all matches of a pattern in a string.
- **preg_replace():** Searches for a pattern in a string and replaces it with a specified value.

Common Regular Expression Patterns for Validation:

- Email addresses: `/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/`
- Phone numbers: `/^\d{3}-\d{3}-\d{4}$/`
- Dates: `/^\d{4}-\d{2}-\d{2}$/`
- URLs: `/^(https?:\/\/)?([a-zA-Z0-9.-]+\.[a-zA-Z]{2,})(\/S*)?$/`

To validate user input, use functions like `preg_match()` to check if a pattern matches the input. If the pattern matches, the input is considered valid; otherwise, it is invalid. Combine regular expressions with other form validation techniques to ensure data integrity.

Example Usage:

```
$email = "example@example.com";
if (preg_match('/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/ ', $email))
{
    echo "Email is valid.";
} else {
    echo "Email is invalid.";
}
```

In the above example, the `preg_match()` function checks if the `$email` variable matches the specified email pattern. If it matches, the email is considered valid and the corresponding message is displayed.

Regular expressions in PHP provides a powerful and flexible way to validate and manipulate strings based on specific patterns. By understanding regular expression syntax and using appropriate patterns, you can enhance your form validation and data processing capabilities in PHP.

Send Values to a Script Manually

To send values to a script manually in PHP, you can use either GET or POST methods. Here's an explanation of both methods:

GET Method:

- The GET method appends data to the URL as query parameters.
- You can **manually** construct a URL with the desired parameters and send a GET request to the PHP script.
- The data is visible in the URL, making it suitable for sending small amounts of data or when you want the data to be bookmarkable or shareable.

In PHP, you can access the values using the `$_GET` superglobal.

Example:

```
<a href="script.php?param1=value1&param2=value2">Send Values</a>
```

In the PHP script (`script.php`):

```
$param1 = $_GET['param1'];  
$param2 = $_GET['param2'];  
  
echo "Parameter 1: " . $param1;  
echo "Parameter 2: " . $param2;
```

POST Method:

- The POST method sends data in the body of the HTTP request, rather than as part of the URL.
- You can manually construct an HTML form with hidden input fields and submit the form programmatically using JavaScript or by clicking a button.
- The data is not visible in the URL, providing better security and privacy.
- In PHP, you can access the values using the `$_POST` superglobal.

Example:

```
<form action="script.php" method="post">
    <input type="hidden" name="param1" value="value1">
    <input type="hidden" name="param2" value="value2">
    <input type="submit" value="Send Values">
</form>
```

In the PHP script (**script.php**):

```
$param1 = $_POST['param1'];
$param2 = $_POST['param2'];

echo "Parameter 1: " . $param1;
echo "Parameter 2: " . $param2;
```

Work with Forms and arrays of data

Working with forms and arrays of data in PHP allows you to handle multiple form inputs efficiently. You can retrieve form data as an array and perform various operations on it. Here's an explanation of how to work with forms and arrays of data in PHP:

Retrieving Form Data as an Array:

- When multiple inputs have the same name attribute, PHP automatically creates an array of values for that input.
- Use square brackets [] in the input name attribute to indicate that the input should be treated as an array.
- In the PHP script, you can access the array of values using the corresponding input name.

```
<form method="post" action="process.php">
    <input type="text" name="name[]" placeholder="Name 1">
    <input type="text" name="name[]" placeholder="Name 2">
    <input type="text" name="name[]" placeholder="Name 3">
    <input type="submit" value="Submit">
</form>
```

In the PHP script (process.php):

```
$names = $_POST['name'];  
foreach ($names as $name) {  
    echo $name . "<br>";  
}
```

The above example will display all the names entered in the form.

Loops in PHP

Loops allow you to execute a block of code repeatedly. The most common loops in PHP are the ***for loop***, ***the while loop***, ***the do-while loop***, and ***foreach loop***.

-
- The ***for loop executes*** a block of code a specified number of times. The basic syntax of a for loop is as follows:

```
for (initialization; condition; increment/decrement) {  
    // execute this code  
}
```

For example:

```
for ($i = 0; $i < 5; $i++) {  
    echo "The value of i is " . $i . "<br>";  
}
```

In the above example, the for loop executes the code block five times, incrementing the variable \$i each time.

-
- The ***while loop executes*** a block of code as long as a certain condition is true. The basic syntax of a while loop is as follows:

```
while (condition) {  
    // execute this code  
}
```

For example:

```
$i = 0;  
while ($i < 5) {  
    echo "The value of i is " . $i . "<br>";  
    $i++;  
}
```

In the above example, the while loop executes the code block as long as the variable \$i is less than 5, incrementing \$i each time.

-
- The ***do-while loop*** is similar to the while loop, but it always executes the code block at least once. The basic syntax of a do-while loop is as follows:

```
do {  
    // execute this code  
} while (condition);
```

For example:

```
$i = 0;  
do {  
    echo "The value of i is " . $i . "<br>";  
    $i++;  
} while ($i < 5);
```

In the above example, the do-while loop executes the code block at least once, and then continues to execute the block as long as the variable \$i is less than 5.

-
- The ***foreach loops*** through a block of code for each element in an array. The basic syntax of a foreach loop is as follows:

```
foreach ($array as $value) {  
    // code to be executed for each iteration  
}
```

The code block within the curly braces {} is the body of the loop. It contains the statements that will be executed for each iteration.

For example:

```
$fruits = ['apple', 'banana', 'orange'];  
foreach ($fruits as $fruit) {  
    echo $fruit . "<br>";  
}
```

In the above example, the foreach loop iterates over each element of the \$fruits array, and in each iteration, the value of the current element is stored in the \$fruit variable. The loop body simply **echoes** the current fruit value.

Files and Directories

PHP provides a variety of functions and features for handling files and directories. Whether you need to check the existence of a file, read from it, write to it, or perform other file-related tasks, PHP offers a comprehensive set of tools for efficient file handling. Working with files is an essential aspect of many web applications, allowing you to store and retrieve data, process user input, and manage resources

PHP include and require Statements

Including files is a common practice in PHP that allows you to reuse code and modularize your application. PHP provides several methods to include files within your scripts, allowing you to import and execute code from external files. Here are the different ways to include files in PHP:

1. include:

- The ``include`` statement is used to include a file in the current script.
- If the included file cannot be found or has errors, a warning is issued, but the script continues to execute.
- Syntax:

```
include 'filename';
```

2. require:

- The ``require`` statement is similar to ``include``, but it generates a fatal error if the included file is not found or has errors.
- Syntax:

```
require 'filename';
```

3. include_once:

- The ``include_once`` statement includes a file only once, even if it is called multiple times in the script.
- It prevents duplicate inclusion of the same file, avoiding naming conflicts and reducing memory usage.
- Syntax:

```
include_once 'filename';
```

4. require_once:

- The ``require_once`` statement is similar to ``include_once``, but it generates a fatal error if the file is not found or has errors.
- It ensures that a file is included only once throughout the script execution.

- Syntax:

```
require_once 'filename';
```

When including files, it is common to use relative paths to specify the file location. The path can be relative to the current script or the server's document root. Here are some examples:

- ✓ *Including a file in the same directory:*

```
include 'file.php';
```

- ✓ *Including a file from a subdirectory:*

```
`include 'subdirectory/file.php';`
```

- ✓ *Including a file using an absolute path:*

```
`include 'C:/xampp/htdocs/html/file.php';`
```

It's important to note that included files have access to the variables and functions in the including script. This allows you to share data and functionality between different parts of your application.

Including files is useful for various purposes, such as:

- Reusing common code across multiple scripts.
- Separating large code blocks into manageable chunks.
- Including configuration files, libraries, or frameworks.
- Modularizing your code for easier maintenance and organization.

By using PHP's file inclusion methods, you can effectively reuse code, improve code structure, and enhance the maintainability of your PHP applications.

Testing for the Existence of Files

Before working with a file, it is often necessary to check if it exists. PHP provides functions such as **file_exists()** and **is_file()** to determine if a file exists in a specified location. These functions return boolean values (true or false) based on the file's presence. By checking for file existence, you can ensure that your code only operates on valid files, avoiding errors and unexpected behavior.

Syntax

```
if (file_exists('test.txt')) {  
    echo "The file exists!";  
}
```

Opening the file

Before you can manipulate the contents of a file in PHP, you need to open it using the **fopen()** function. This function takes two parameters: the file path and the mode in which the file should be opened.

The **fopen()** function is used to establish a connection between your PHP script and the file on the server. It returns a file handle, which is a special resource that represents the opened file. This file handle is used in subsequent file-related operations like reading, writing, or closing the file.

The first parameter of **fopen()** is the file path, which specifies the location of the file you want to open. It can be an absolute path (e.g., **/path/to/file.txt**) or a relative path to the script executing the function. Relative paths are resolved relative to the script's location.

The second parameter is the mode in which the file should be opened. The mode determines whether you want to read, write, or append to the file, among other options. Here are some common file modes:

1. Read mode (r):
 - Open the file for reading only.
 - The file pointer is positioned at the beginning of the file.
 - If the file does not exist, an error is generated.
2. Write mode (w):
 - Open the file for writing only.
 - If the file does not exist, it will be created.
 - If the file exists, its contents are truncated (emptied) before writing.
 - The file pointer is positioned at the beginning of the file.
3. Append mode (a):
 - Open the file for writing only.
 - If the file does not exist, it will be created.
 - If the file exists, the file pointer is positioned at the end of the file, allowing you to append data without overwriting existing content.

Other file modes include:

- Read and write (r+): Open the file for reading and writing.
- Write and create (w+): Open the file for reading and writing, truncating the file or creating it if it doesn't exist.
- Append and create (a+): Open the file for reading and writing, positioning the file pointer at the end of the file.

MODE	READ	WRITE	CREATE FILE*	NEW	TRUNCATE
r	1	0	0		0
w	0	1	1		1
a	0	1	1		0
r+	1	1	0		0
w+	1	1	1		1
a+	1	1	1		0

* Create New File if it doesn't exist

Example usage of **fopen()** to open a file for reading:

```
$file = fopen('path/to/file.txt', 'r');
if ($file) {
    // Perform operations on the opened file
    // ...
    fclose($file); // Close the file handle
} else {
    // Handle the case when the file cannot be opened
    echo "Failed to open the file.";
}
```

After opening the file using **fopen()**, you can perform various operations like reading from the file using functions like **fread()**, writing to the file using functions like **fwrite()**, or navigating within the file using functions like **fseek()**. Once you are done working with the file, it's essential to close the file handle using the **fclose()** function to release system resources.

In summary, the **fopen()** function in PHP allows you to open files for reading, writing, or both. It takes a file path and a mode parameter to specify the desired file access. Understanding file modes and using **fopen()** correctly enables you to manipulate file contents in PHP applications effectively.

Read from Files

PHP offers functions like **file_get_contents()** and **fread()** to read the contents of a file. These functions allow you to retrieve the entire file content or read it line by line. Reading from files is useful when you need to process data stored in a file, such as reading configuration files, parsing log files, or retrieving user-uploaded files for further processing.

In PHP, you can read data from files using various functions that allow you to read by the byte, by the line, or even by the single character. Two commonly used functions for reading lines from a file are **fgets()** and **feof()**.

The **fgets()** function is used to read a line from an open file. It takes the file resource returned by the **fopen()** function as its argument. Each time **fgets()** is called, it reads a single line from the file and moves the file pointer to the next line. The function returns the line as a string, including the newline character at the end.

Here's an example of using **fgets()** to read lines from a file:

```
$file = fopen('path/to/file.txt', 'r');
if ($file) {
    while (($line = fgets($file)) !== false) {
        // Process the line
        echo $line;
    }
    fclose($file);
} else {
    echo "Failed to open the file.";
}
```

In this example, the **fgets()** function is called inside a loop that continues until **fgets()** returns false, indicating that the end of the file has been reached. The loop reads each line from the file and processes it as needed. It's important to note that the newline character at the end of each line is included in the returned string.

To determine if you have reached the end of the file, you can use the **feof()** function. It takes the file resource as its argument and returns true if the end of the file has been reached, or false otherwise.

Here's an example of using **feof()** to read lines from a file until the end:

```
$file = fopen('path/to/file.txt', 'r');
if ($file) {
    while (!feof($file)) {
        $line = fgets($file);
        // Process the line
        echo $line;
    }
    fclose($file);
} else {
    echo "Failed to open the file.";
}
```

In this example, the **feof()** function is used as the loop condition. The loop continues until **feof()** returns true, indicating that the end of the file has been reached. Inside the loop, **fgets()** is used to read each line from the file, and the line is processed accordingly.

By using functions like **fgets()** and **feof()**, you can efficiently read lines from files in PHP, allowing you to process file content line by line, perform specific operations, or extract data for further manipulation.

In PHP, there is also another option to read a file in chunks or specified sizes rather than reading it line by line. The **fread()** function allows you to read a specific number of bytes from an open file.

The **fread()** function takes two parameters: the file resource returned by the **fopen()** function and the number of bytes you want to read from the file. It reads the specified number of bytes from the current position of the file pointer and advances the pointer accordingly. The function returns the data read as a string.

Here's an example of using **fread()** to read a file in chunks:

```
$file = fopen('path/to/file.txt', 'r');
if ($file) {
    $chunkSize = 1024; // Specify the chunk size in bytes

    while (true) {
        $data = fread($file, $chunkSize);
        // Process the data
        echo $data;
        if ($data === false) break;
        fclose($file);
    }
} else {
    echo "Failed to open the file.";
}
```

In this example, the **fread()** function is used inside a loop to read the file in chunks. The chunk size is defined using the variable **\$chunkSize**, which specifies the number of bytes to be read in each iteration. The loop continues until the end of the file is reached, as determined by the **feof()** function.

The **fread()** function will return the amount of data requested (i.e., the chunk size) unless the end of the file is encountered before reaching the requested size. In that case, it will return the remaining data available in the file. This allows you to handle files of various sizes efficiently.

By using **fread()** to read files in chunks, you can control the amount of data read at a time, which can be useful when dealing with large files or when you need to process data in specific-sized segments. This approach gives you more flexibility in handling file contents according to your specific requirements.

Write to Files

In PHP, you can write data to a file or append text to an existing file using the **fwrite()** function. This function allows you to write data to a specified file pointer.

The **fwrite()** function requires two main arguments: the file pointer and the string of data to be written. The file pointer represents the open file where you want to write the data. It is obtained by using the **fopen()** function with the appropriate file path and mode.

Here's an example of using **fwrite()** to write data to a file:

```
$file = fopen('path/to/file.txt', 'w');
if ($file) {
    $data = "This is some data to write to the file.";
    fwrite($file, $data);
    fclose($file);
    echo "Data written successfully.";
} else {
    echo "Failed to open the file.";
}
```

In this example, the **fwrite()** function is used to write the string **\$data** to the file specified by the file pointer **\$file**. The file is opened in write mode ('w') using **fopen()**. After writing the data, the file is closed with **fclose()**.

You can also specify an optional third argument in **fwrite()** to limit the length of the data to be written. If you include a length argument, writing will stop after the specified length has been reached. This can be useful when you want to write only a portion of a larger string or when you need to control the amount of data written.

Here's an example of using **fwrite()** with a specified length:

```
$file = fopen('path/to/file.txt', 'w');
if ($file) {
    $data = "This is some data to write to the file.";
    $length = 10; // Write only the first 10 characters
    fwrite($file, $data, $length);
    fclose($file);
    echo "Data written successfully.";
} else {
    echo "Failed to open the file.";
}
```

In this example, **fwrite()** will write only the first 10 characters of the string **\$data** to the file.

By using the **fwrite()** function, you can easily write data to a file or append text to an existing file in PHP. It provides flexibility in controlling the length of data written and allows you to handle file operations effectively in your web applications.

Working with directories

PHP provides a rich set of functions for working with directories, allowing you to perform various operations such as creating, reading, modifying, and deleting directories. These functions provide flexibility and control when it comes to managing directory structures within your PHP applications. Here are some of the key functions for directory handling in PHP:

1. **mkdir()**: This function is used to create a new directory. It takes the directory path as a parameter and can also accept optional parameters to specify permissions and whether to create nested directories if they don't already exist.
2. **rmdir()**: This function is used to remove a directory. It takes the directory path as a parameter and deletes the specified directory. Note that the directory must be empty for the function to succeed.
3. **is_dir()**: This function is used to check if a given path is a directory. It returns a boolean value (**true** if it's a directory, **false** otherwise) based on whether the specified path points to a directory or not.
4. **opendir()**: This function is used to open a directory and obtain a directory handle. It takes the directory path as a parameter and returns a directory handle resource that can be used to read the contents of the directory.
5. **readdir()**: This function is used to read the contents of a directory. It takes a directory handle resource obtained from **opendir()** as a parameter and returns the name of the next file or directory in the directory. It can be called in a loop to iterate over all the items in the directory.
6. **closedir()**: This function is used to close a directory handle. It takes a directory handle resource obtained from **opendir()** as a parameter and releases the resources associated with the directory handle.

These are just a few examples of the functions available for directory handling in PHP. There are additional functions like **scandir()**, **chdir()**, **getcwd()**, and many more that provide further functionality for working with directories.

By utilizing these directory functions, you can effectively manage directory structures, create new directories, check their existence, read their contents, and perform various other operations to organize and manipulate files and directories within your PHP applications.

Example:

```
// Create a new directory
$directory = 'path/to/new_directory';
mkdir($directory);

// Check if the directory exists
if (is_dir($directory)) {
    echo "Directory '{$directory}' exists.<br>";

    // Open the directory handle
    $handle = opendir($directory);
    if ($handle) {
        echo "Files in '{$directory}':<br>";

        // Read and display the files in the directory
        while (($file = readdir($handle)) !== false) {
            if ($file != '.' && $file != '..') {
                echo "- {$file}<br>";
            }
        }

        // Close the directory handle
        closedir($handle);
    } else {
        echo "Failed to open the directory.";
    }
} else {
    echo "Directory '{$directory}' does not exist.";
}

// Remove the directory
rmdir($directory);
```

In this example, we first create a new directory using the **mkdir()** function. We then use the **is_dir()** function to check if the directory exists. If it exists, we open the directory handle using **opendir()** and iterate over the files in the directory using **readdir()**. We skip the `.` and `..` entries, which represent the current directory and parent directory respectively. Finally, we close the directory handle with **closedir()**.

After performing the necessary operations on the directory, we remove it using the **rmdir()** function.

Note that you need to replace **'path/to/new_directory'** with the actual directory path you want to work with.

This example demonstrates how you can create a directory, check its existence, read and display the files within it, and remove the directory using PHP's directory functions.

Upload Files

When it comes to file uploading in PHP, you can use a combination of HTML forms and PHP scripts to allow users to upload files to the server. Here's an overview of the process:

1. HTML Form Setup:

- Create an HTML form with the **enctype** attribute set to "**multipart/form-data**". This attribute is required for file uploads.
- Include an **<input type="file">** element in the form to allow users to select the file they want to upload.
- Add any additional form fields you need for capturing related information.

2. PHP Script for File Upload:

- In the PHP script that processes the form submission, check if a file has been uploaded using the **\$_FILES** superglobal.
- Access the uploaded file information through **\$_FILES['fileFieldName']**, where **'fileFieldName'** is the name of the file input field in your form.
- Use **move_uploaded_file()** function to move the uploaded file from the temporary directory to the desired destination on the server. Specify the source file path and the destination path as arguments.

3. Handling File Upload Errors:

- Check for any errors during the file upload process using the **\$_FILES['fileFieldName']['error']** variable. This variable will contain an error code if an issue occurs.
- Display appropriate error messages or take necessary actions based on the error code.

Regarding the temporary directory and file size limits, you can find the relevant information in the **phpinfo.php** page. The **upload_tmp_dir** parameter specifies the temporary directory where uploaded files are initially stored before processing. The **upload_max_filesize** parameter defines the maximum allowed size of uploaded files.

To modify these parameters, you can access the **php.ini** configuration file. Locate the **upload_tmp_dir** directive to change the temporary directory path, and adjust the **upload_max_filesize** directive to set the maximum file size limit.

It's important to note that changing these values may require server configuration changes and restarting the PHP service for the modifications to take effect.

By following these steps and considering the temporary directory and file size limitations, you can implement file uploading functionality in your PHP applications.

Certainly! Here's an example of a PHP script that handles file uploading:

HTML Form (upload.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>File Upload Form</title>
</head>
<body>
    <h2>File Upload Form</h2>
    <form action="upload.php" method="POST" enctype="multipart/form-
data">
        <input type="file" name="fileToUpload">
        <input type="submit" value="Upload">
    </form>
</body>
</html>
```

PHP Script (upload.php):

```
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Check if file was uploaded without errors
    if(isset($_FILES['fileToUpload']) && $_FILES['fileToUpload']['error']
=== UPLOAD_ERR_OK) {
        $targetDir = 'uploads/'; // Directory to store uploaded files
        $targetFile = .
        basename($_FILES['fileToUpload']['name']);

        // Move the uploaded file to the desired destination
        if (move_uploaded_file($_FILES['fileToUpload']['tmp_name'],
$targetFile)) {
            echo "File uploaded successfully.";
        } else {
            echo "Error uploading the file.";
        }
    } else {
        echo "Error: “. $_FILES['fileToUpload']['error'];
    }
}
?>
```

In this example, we have an HTML form with an input field of type "file" that allows users to select a file for uploading. The form's action attribute is set to "upload.php" to submit the form data to the PHP script for processing. The form also has the "enctype" attribute set to "multipart/form-data" to enable file uploads.

The PHP script (**upload.php**) checks if the form was submitted via POST request. It then checks if a file was uploaded without any errors by accessing **\$_FILES['fileToUpload']['error']**. If there are no errors, it specifies the target directory (**uploads/** in this case) and the target file path using **basename()** function. The **move_uploaded_file()** function is used to move the uploaded file from the temporary directory to the specified destination.

If the file is successfully uploaded, it displays a success message. Otherwise, it displays an error message.

Make sure to create a directory named "uploads" in the same directory as the PHP script to store the uploaded files.

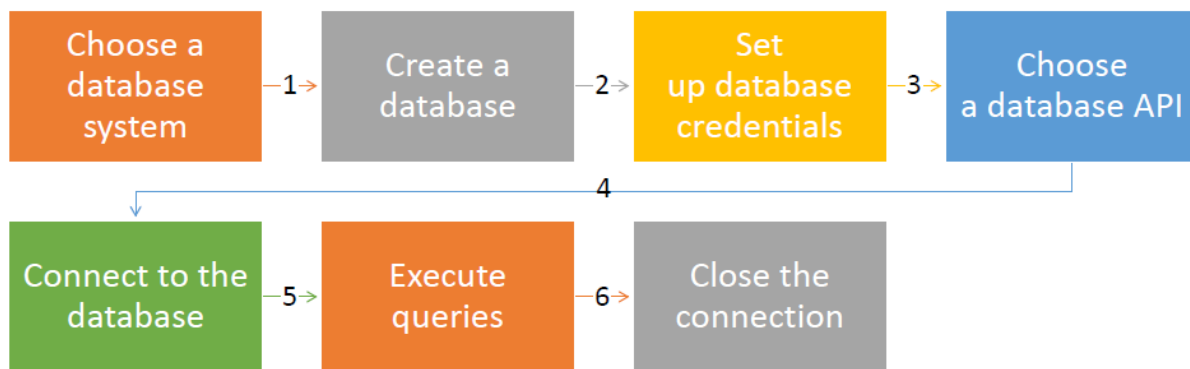
This example demonstrates a basic implementation of file uploading in PHP. You can customize it further based on your specific requirements, such as adding additional form fields, validating file types, or handling file name conflicts.

Manipulating Databases with PHP

Server-side applications require a reliable and efficient way to manage data, and databases are commonly used for this purpose. PHP, being a popular server-side programming language, provides built-in support for working with databases. This allows developers to easily store, retrieve, and manipulate data within their applications.

PHP supports various database systems, including MySQL, PostgreSQL, SQLite, and more. These systems have their own unique features and capabilities, but PHP provides a consistent interface to interact with them regardless of the underlying database.

To establish a connection to a database in PHP, you have two main options: the `mysqli` extension and the `PDO` extension.



Connecting to a MySQL Database

1. **mysqli Extension:** The `mysqli` extension, which stands for "MySQL improved", is specifically designed for MySQL databases. It offers both procedural and object-oriented approaches to interact with MySQL. The `mysqli` extension provides a rich set of functions and features for executing SQL queries, managing transactions, handling errors, and retrieving data from the database. It also supports prepared statements, which help prevent SQL injection attacks.


```
<?php
$servername = "localhost";
$username = "root";
$password = "password";
$dbname = "mydatabase";

// Create a connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

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

echo "Connected successfully";

// Perform database operations here...

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

In this example, we specify the server name, username, password, and database name to establish a connection using **mysqli_connect()**. If the connection is successful, we echo a "Connected successfully" message. After performing the required database operations, we close the connection using **mysqli_close()**.

2. **PDO Extension:** The PDO (PHP Data Objects) extension is a flexible and consistent database abstraction layer in PHP. It supports multiple database systems, allowing you to switch between different databases without changing much of your code.

Example of connecting to a MySQL database using PDO:

```
<?php
$servername = "localhost";
$username = "root";
$password = "password";
$dbname = "mydatabase";

try {
    // Create a connection
    $conn = new PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);

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

    echo "Connected successfully";

    // Perform database operations here...

    // Close the connection (optional, as PDO automatically closes it
when the script ends)
    $conn = null;
} catch (PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

In this example, we use the **PDO** class to establish a connection to the MySQL database. We specify the server name, username, password, and database name in the connection string. The **setAttribute()** method is used to set the error mode to exception, which throws an exception if any errors occur during database operations. After performing the required database operations, the connection is automatically closed or you can explicitly close it using **\$conn = null;**.

By using either the **mysqli** or PDO extension, you can connect to the desired database system and perform various database operations such as querying data, inserting records, updating data, and deleting records. These extensions provide a wide range of functions and methods to handle database connectivity effectively and securely in PHP.

Connecting to a MySQL Database:

To connect to a MySQL database using PHP, you need to use the **mysqli** or PDO extension. Here's an example using the **mysqli** extension:

```
// Database configuration
$host = 'localhost';
$user = 'username';
$password = 'password';
$database = 'database';

// Create a new mysqli object
$mysqli = new mysqli($host, $user, $password, $database);

// Check for errors
if ($mysqli->connect_error) {
    die('Connect Error ( ' . $mysqli->connect_errno . ' ) ' . $mysqli->connect_error);
}
echo 'Connected successfully!';
```

In the above example, we first define the database configuration variables (\$host, \$user, \$password, and \$database). Then, we create a new mysqli object and pass in the database configuration variables. If there's an error, we use the die() function to display an error message.

TASKS	MYSQLI	PDO
Connect to the db	<code>\$c=new mysqli(\$host, \$user, \$pass, \$dbname);</code>	<code>\$c=new PDO("mysql:host=\$host;dbname=myDB", \$user, \$pass);</code>
Execute queries	<code>\$result = \$c->query(\$sql);</code>	<code>\$c->exec(\$sql); //insert,update, delete</code> <code>\$s=\$c->query(\$sql); // select</code>
Fetch result	<pre>while(\$row = \$result->fetch_assoc()) { echo "id: " . \$row["id"]. " - UserName: " . \$row["username"]. " - Email: " . \$row["email"]. "
"; }</pre>	<pre>\$s->setFetchMode(PDO::FETCH_ASSOC); while(\$row = \$s->fetch()) { echo "id:".\$row["id"]."- UserName:".\$row["username"]."- Email:".\$row["email"]."
"; }</pre>
Close Connection	<code>\$c->close();</code>	<code>\$c=null;</code>

Sending Data to a MySQL Database:

To send data to a MySQL database using PHP, you need to use SQL statements such as INSERT.

Here's an example:

```
// SQL query to insert data
$sql = "INSERT INTO users (name, email) VALUES ('John Doe', 'johndoe@example.com')";

// Execute the query
if ($mysqli->query($sql) === TRUE) {
    echo 'Data inserted successfully!';
} else {
    echo 'Error inserting data: ' . $mysqli->error;
}
```

In the above example, we use the INSERT statement to insert data into the users table. We execute the query using the query() method of the mysqli object. If there's an error, we display an error message.

Retrieving Data from a MySQL Database

To retrieve data from a MySQL database using PHP, you need to use SQL statements such as SELECT. Here's an example:

```
// SQL query to select data
$sql = "SELECT * FROM users";
// Execute the query and fetch the results
$result = $mysqli->query($sql);
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo 'Name: ' . $row['name'] . '<br>';
        echo 'Email: ' . $row['email'] . '<br><br>';
    }
} else {
    echo 'No data found!';
}
```

In the above example, we use the SELECT statement to select all data from the users table. We execute the query using the query() method of the mysqli object, and then loop through the results using a while loop. We use the fetch_assoc() method to fetch each row of data as an associative array, and then display the data on the screen.

Modifying and Removing Data from a MySQL Database:

To modify or remove data from a MySQL database using PHP, you need to use SQL statements such as UPDATE and DELETE. Here's an example:

```
// SQL query to update data

$sql = "UPDATE users SET email='johndoe_updated@example.com' WHERE id=1";


// Execute the query
if ($mysqli->query($sql) === TRUE) {
    echo 'Data updated successfully!';
} else {
    echo 'Error updating data: ' . $mysqli->error;
}
```

Introduction to PHP Sessions and Cookies

Sessions and cookies are mechanisms used in web development to maintain and manage user data. Sessions are server-side storage areas where data can be stored and accessed throughout a user's browsing session. Cookies are small text files stored on the client-side that contain data sent by the server and are used for tracking and identifying users.

Sessions and cookies play a crucial role in maintaining user state and personalization on websites. They allow websites to remember user preferences, login information, and other data across different pages and sessions. Sessions and cookies enable features such as shopping carts, user authentication, and personalized experiences. They enhance user convenience and improve the overall user experience on websites.

Understanding statelessness and the need for session management

HTTP is a stateless protocol, meaning it doesn't inherently maintain information about previous requests. Session management is necessary to overcome the statelessness of HTTP and provide a seamless browsing experience. Sessions enable the server to associate data with a specific user across multiple requests and responses. They allow websites to remember user interactions, track progress, and provide customized content. Session management helps in maintaining user authentication, preventing unauthorized access, and securing sensitive information.

By utilizing sessions and cookies, web developers can create dynamic and interactive websites that remember user preferences, maintain state, and provide personalized experiences. Understanding the concepts and importance of sessions and cookies is crucial for effective web development and user engagement.

PHP Session

What are sessions?

- Sessions are a way to store and manage data on the server side for individual users.
- A session starts when a user accesses a website and ends when the user closes the browser or the session expires.
- Sessions allow for persistent data storage and retrieval throughout a user's browsing session.

How sessions work in PHP:

- When a session is started, a unique session ID is generated for the user and stored as a cookie or passed through the URL.

- The session ID allows the server to identify the user and retrieve the corresponding session data.
- Session data is stored on the server, typically in a temporary directory or a database.
- PHP provides built-in functions and mechanisms to manage sessions effectively.

Starting a session:

1. Using session_start() function:

- To start a session in PHP, the session_start() function is used at the beginning of the PHP script.
- It initializes the session and makes the session data available for use.

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

2. Configuring session settings:

- PHP provides various configuration options for sessions, such as session storage, session lifetime, and session handling.
- These settings can be modified in the php.ini file or using the ini_set() function within the script.

C. Storing and accessing session data:

1. Storing data in session variables:

- Session variables are used to store data that needs to be accessed across multiple pages within the same session.
- Data can be stored in session variables using the \$_SESSION superglobal array.

```
<?php
    // After successful login, store user data in session variables
    $_SESSION['username'] = 'JohnDoe';
    $_SESSION['email'] = 'johndoe@example.com';
?>
```

2. Accessing session data across multiple pages:

- Once session data is stored, it can be accessed on different pages within the same session.
- By referencing the session variables using the `$_SESSION` array, the stored data can be retrieved and utilized.

```
<?php
    // Retrieve and display user data from session variables
    echo 'Welcome, ' . $_SESSION['username'];
    echo 'Your email address is: ' . $_SESSION['email'];
?>
```

D. Modifying and deleting session data:

1. Updating session variables:

- Session variables can be updated by assigning new values to them using the `$_SESSION` array.

```
<?php
    // Update the email address in the session
    $_SESSION['email'] = 'newemail@example.com';
?>
```

2. Removing session variables:

- Session variables can be unset or removed using the `unset()` function.

```
<?php
    // Remove the email from the session
    unset($_SESSION['email']);
?>
```


3. Destroying a session:

- To end a session and remove all session data, the `session_destroy()` function is used.
- This terminates the session and deletes the session file from the server.

```
<?php
    // Log out and destroy the session
    session_destroy();
?>
```

In this example, when a user logs in, their username and email are stored in session variables. These variables can be accessed and used on the profile page or any other pages within the same session. The user's email can be updated or removed from the session as needed. Finally, when the user logs out, the session is destroyed, and all session data is removed.

PHP Cookies

What are cookies?

1. Definition and purpose of cookies: Cookies are small text files stored on the user's computer by the web server. They are used to store user-specific information and maintain state between multiple requests.
2. How cookies work in PHP: When a user visits a website, the server sends a Set-Cookie header with the response, which instructs the browser to store the cookie. The browser then includes the cookie in subsequent requests to the same server.

B. Setting and retrieving cookies

1. Using `setcookie()` function: The `setcookie()` function is used to set a cookie. It takes parameters such as the name, value, expiration time, path, domain, and secure flag.

```
<?php
    setcookie('username', 'JohnDoe', time() + 3600);

    // Set a cookie with the name 'username' and value 'JohnDoe' that
    expires in 1 hour
?>
```

2. Configuring cookie settings: You can specify additional settings when setting a cookie, such as the expiration time, path, domain, and secure flag.

```
<?php
    setcookie('username', 'JohnDoe', time() + 3600, '/',
    'example.com', true); // Set a cookie with additional settings
?>
```

C. Reading and manipulating cookie values

1. Accessing cookie values: To retrieve the value of a cookie, you can use the `$_COOKIE` superglobal array.

```
<?php
    echo $_COOKIE['username']; // Output the value of the 'username'
    cookie
?>
```

2. Updating and deleting cookies: To update a cookie, you can simply set a new value using the `setcookie()` function. To delete a cookie, you can set its expiration time to a past date.

```
<?php
    setcookie('username', 'JaneDoe', time() + 3600);
    // Update the value of the 'username' cookie
    setcookie('username', '', time() - 3600);
    // Delete the 'username' cookie by setting its expiration time
    to the past
?>
```

PHP cookies allow you to store and retrieve user-specific information on the client-side. They are useful for maintaining user sessions, remembering user preferences, and implementing personalized features. By using the `setcookie()` function and accessing the `$_COOKIE` superglobal, you can easily work with cookies in PHP.

References

1. PHP Documentation: The official documentation for PHP provides comprehensive information on the PHP language, including PHP forms, file handling, database connectivity, and more. You can access it at: <https://www.php.net/docs.php>
2. HTML Forms: The Mozilla Developer Network (MDN) provides detailed documentation and examples on HTML forms. You can find more information at: <https://developer.mozilla.org/en-US/docs/Learn/Forms>
3. PHP Form Validation: The W3Schools website offers tutorials on PHP form validation. You can visit their PHP Form Validation page at: https://www.w3schools.com/php/php_form_validation.asp
4. Regular Expressions in PHP: The PHP documentation includes a section specifically dedicated to regular expressions. You can find more details and examples at: <https://www.php.net/manual/en/book.regex.php>
5. File Handling in PHP: The PHP documentation provides a complete reference for working with files in PHP. You can refer to the official documentation at: <https://www.php.net/manual/en/ref.filesystem.php>
6. PHP Database Connectivity:
 - For mysqli extension: The PHP documentation provides a detailed guide on using mysqli for database connectivity. You can find it at: <https://www.php.net/manual/en/book.mysqli.php>
 - For PDO extension: The PHP documentation also offers a guide on using PDO for database connectivity. You can access it at: <https://www.php.net/manual/en/book.pdo.php>
7. PHP Sessions and Cookies:
 - PHP Manual - Sessions: <https://www.php.net/manual/en/book.session.php>
 - PHP Manual - Cookies: <https://www.php.net/manual/en/features.cookies.php>
 - W3Schools - PHP Sessions: https://www.w3schools.com/php/php_sessions.asp

Model Questions & Answers

Questions

1. What is the advantage of server-side scripting over client-side scripting?
 - a) Faster execution
 - b) Improved security
 - c) Better user experience
 - d) Platform independence
2. Which of the following is a server-side scripting language?
 - a) JavaScript
 - b) HTML
 - c) PHP
 - d) CSS
3. How can you include PHP code in an HTML document?
 - a) Using the <script> tag
 - b) Using the <php> tag
 - c) Using the <code> tag
 - d) Using the <?php> tag
4. What is the purpose of PHP form validation?
 - a) To improve website performance
 - b) To protect against SQL injection attacks
 - c) To ensure proper data entry by users
 - d) To enhance user interface design
5. Which PHP function is used to read data from files?
 - a) fopen()
 - b) fwrite()
 - c) feof()
 - d) fgets()
6. How can you send data to a PHP script manually?
 - a) Using the GET method
 - b) Using the POST method
 - c) Using the PUT method
 - d) Using the DELETE method
7. What is the purpose of PHP sessions?
 - a) To store data on the client-side
 - b) To store data on the server-side
 - c) To encrypt data during transmission
 - d) To authenticate user credentials
8. How can you access form data in PHP?
 - a) Using the \$_GET superglobal
 - b) Using the \$_POST superglobal
 - c) Using the \$_REQUEST superglobal
 - d) Using the \$_SESSION superglobal

9. How can you include external PHP files in your script?
 - a) Using the include_once() function
 - b) Using the require_once() function
 - c) Using the include() function
 - d) Using the require() function
10. What is the purpose of PHP cookies?
 - a) To store user preferences on the server-side
 - b) To store user information on the client-side
 - c) To encrypt sensitive data during transmission
 - d) To validate user input in web form
11. How can you declare and initialize a variable in PHP?
 - a) var \$x = 10;
 - b) int \$x = 10;
 - c) \$x = 10;
 - d) declare \$x = 10;
12. What is the purpose of concatenating variables and strings in PHP?
 - a) To convert variables into strings
 - b) To perform mathematical operations
 - c) To combine multiple variables into a single string
 - d) To compare variables with strings
13. Which PHP function is used to remove elements from an array?
 - a) array_remove()
 - b) array_delete()
 - c) array_pop()
 - d) array_unset()
14. How can you open and read the content of a file in PHP?
 - a) fopen() and fread()
 - b) file_open() and file_read()
 - c) open_file() and read_file()
 - d) load_file() and retrieve_file()
15. How can you connect to a MySQL database in PHP?
 - a) Using the mysqli_connect() function
 - b) Using the pdo_connect() function
 - c) Using the mysql_connect() function
 - d) Using the db_connect() function
16. What is the purpose of PHP sessions in managing user state?
 - a) To store user credentials securely
 - b) To track user activity on the website
 - c) To maintain user-specific data across multiple requests
 - d) To cache website content for faster performance
17. How can you set a cookie in PHP?
 - a) set_cookie()
 - b) create_cookie()
 - c) store_cookie()
 - d) setcookie()

18. What is the default lifespan of a PHP session?
- a) Until the browser is closed
 - b) 1 hour
 - c) 24 hours
 - d) 7 days
19. Which function is used to destroy a PHP session?
- a) session_destroy()
 - b) session_remove()
 - c) session_close()
 - d) session_end()
20. How can you prevent session hijacking in PHP?
- a) Using secure HTTPS connections
 - b) Setting session timeouts
 - c) Regenerating session IDs
 - d) Encrypting session data
21. How can you process form data in PHP?
- a) Using the \$_GET superglobal
 - b) Using the \$_POST superglobal
 - c) Using the \$_REQUEST superglobal
 - d) Using the \$_FORM superglobal
22. Which statement is used to create a loop that executes a block of code as long as a certain condition is true?
- a) for
 - b) while
 - c) foreach
 - d) do...while
23. What is the purpose of the include and require statements in PHP?
- a) To include external CSS files
 - b) To import PHP code from other files
 - c) To load JavaScript libraries
 - d) To embed images in the HTML document
24. How can you retrieve data from a MySQL database in PHP?
- a) Using the mysqli_fetch_array() function
 - b) Using the mysql_fetch_row() function
 - c) Using the pdo_fetch() function
 - d) Using the db_fetch() function
25. What is the purpose of the filter_var() function in PHP?
- a) To validate and sanitize user input
 - b) To perform mathematical calculations
 - c) To manipulate strings
 - d) To encrypt data for secure transmission
26. How can you upload files in PHP?
- a) Using the file_upload() function
 - b) Using the file_save() function
 - c) Using the file_transfer() function
 - d) Using the move_uploaded_file() function
27. What is the significance of session management in web applications?
- a) It ensures data persistence between page visits
 - b) It allows users to bookmark specific pages
 - c) It provides secure communication between the client and server
 - d) It enables personalized user experiences and tracking

28. How can you access the value of a cookie in PHP?
- a) `$_COOKIE['cookie_name']`
 - b) `$_SESSION['cookie_name']`
 - c) `$_REQUEST['cookie_name']`
 - d) `$_POST['cookie_name']`
29. What is the purpose of the PDO extension in PHP?
- a) To connect to a database and perform database operations
 - b) To manipulate images and graphics
 - c) To generate dynamic HTML content
 - d) To handle user authentication and authorization
30. What is the difference between `$_GET` and `$_POST` superglobals in PHP?
- a) `$_GET` is used for retrieving form data, while `$_POST` is used for sending data to the server.
 - b) `$_GET` is more secure than `$_POST` for handling sensitive information.
 - c) `$_GET` stores data in the URL, while `$_POST` sends data in the request body.
 - d) `$_GET` can only handle text data, while `$_POST` can handle various data types.

Answer keys

Question	Answer	Question	Answer
1	b) Improved security	16	c) To maintain user-specific data across multiple requests
2	c) PHP	17	d) <code>setcookie()</code>
3	d) Using the <code><?php></code> tag	18	a) Until the browser is closed
4	c) To ensure proper data entry by users	19	a) <code>session_destroy()</code>
5	d) <code>fgets()</code>	20	c) Regenerating session IDs
6	a) Using the GET method	21	b) Using the <code>\$_POST</code> superglobal
7	b) To store data on the server-side	22	b) while
8	b) Using the <code>\$_POST</code> superglobal	23	b) To import PHP code from other files
9	a) Using the <code>include_once()</code> function	24	a) Using the <code>mysqli_fetch_array()</code> function
10	b) To store user information on the client-side	25	a) To validate and sanitize user input
11	c) <code>\$x = 10;</code>	26	d) Using the <code>move_uploaded_file()</code> function
12	c) To combine multiple variables into a single string	27	d) It enables personalized user experiences and tracking
13	c) <code>array_pop()</code>	28	a) <code>\$_COOKIE['cookie_name']</code>
14	a) <code>fopen()</code> and <code>fread()</code>	29	a) To connect to a database and perform database operations
15	a) Using the <code>mysqli_connect()</code> function	30	c) <code>\$_GET</code> stores data in the URL, while <code>\$_POST</code> sends data in the request body.