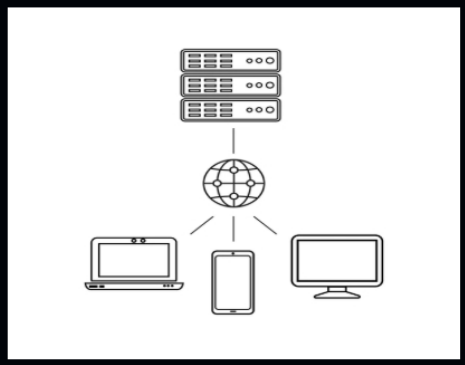
PHP Basics: Introduction to PHP

What is PHP?

PHP (Hypertext Preprocessor) is a popular general-purpose scripting language that is especially suited to web development. It is fast, flexible, and pragmatic. PHP scripts are executed on the server, and the result is sent to the client as plain HTML.

- **Server-side language:** PHP is executed on the server before the page is sent to the user's browser.
- **Open-source:** PHP is free to use and has a large community of developers.
- **Cross-platform:** PHP runs on various platforms, including Windows, Linux, and macOS.



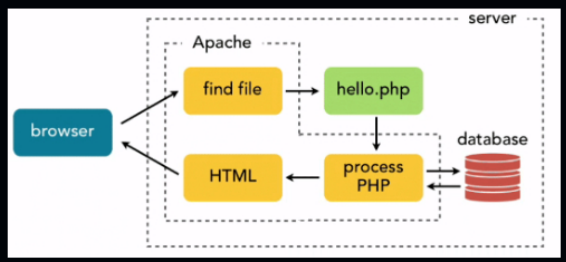
Key Features of PHP

1. Easy to embed into HTML.
2. Supports a wide range of databases (MySQL, PostgreSQL, etc.).
3. Efficient for building dynamic web pages.
4. Powerful in managing forms, user sessions, cookies, and more.

Example: Basic PHP Syntax

The following example demonstrates how to create a simple "Hello, World!" application using PHP:

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



Diff b/w `echo` , `print` and `print_r` functions

- `echo` is used to output one or more strings.
- `print` is used to output a string and automatically appends a newline at the end.
- `print_r` is used to output a string and automatically appends a newline at the end

Setting up `XAMPP/WAMP` for PHP Development

To run PHP code locally, you need to set up a local server environment using tools like XAMPP or WAMP. These tools bundle Apache (server), MySQL (database), and PHP together.

Installing XAMPP

Download XAMPP from https://www.apachefriends.org/index.html.

Install XAMPP and start the Apache server.

Once installed, you can run your PHP scripts from the `htdocs` folder in XAMPP.

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

Save this code in a file named `test.php` in the `htdocs` folder.

Open a web browser and navigate to `http://localhost/test.php` to see the output.

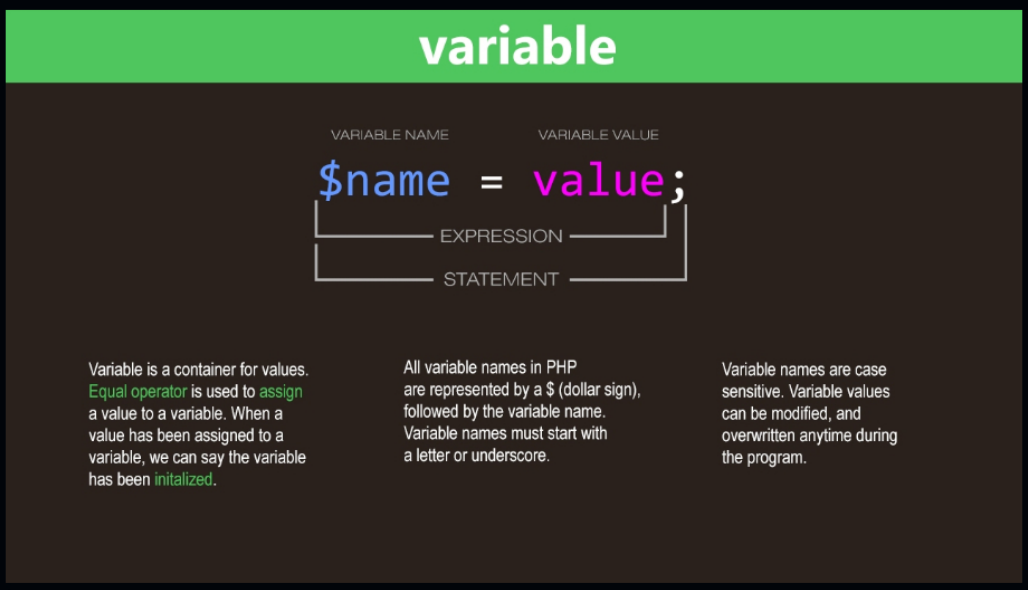
PHP Data Types and Variables

PHP Variables

Variables in PHP are used to store data. A variable starts with the `$` sign, followed by the name of the variable.

Syntax

```php
<?php
  $variableName = value;
?>
```



Example: Using Variables

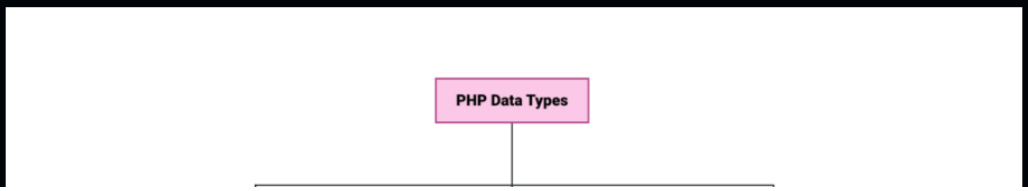
```php
<?php
  $greeting = "Hello, World!";
  $number = 10;

  echo $greeting; // Output: Hello, World!
  echo $number;   // Output: 10
?>
```

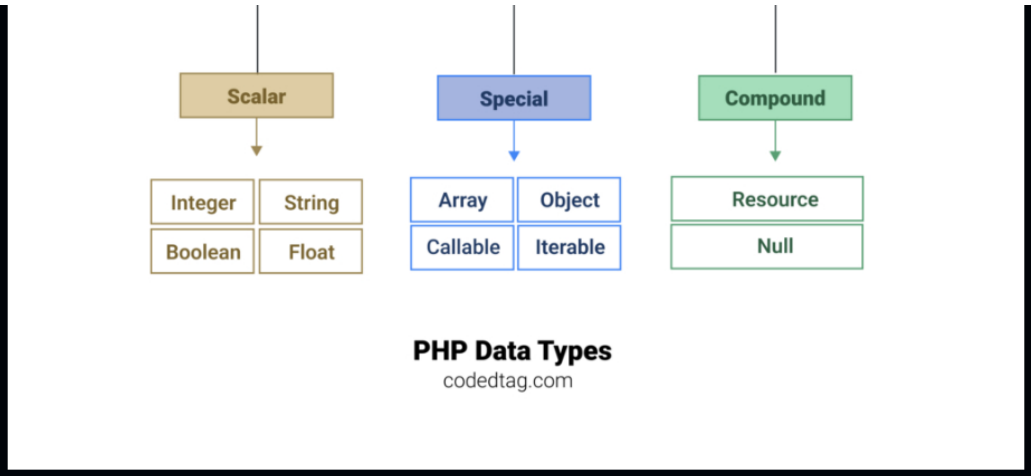
In PHP:

1. Variables must start with a letter or underscore.
2. Variable names are case-sensitive.
3. Variable not start with number after `$` symbol
4. Variable name is combination of a-z , A-Z , 0-9 and _
5. You do not need to declare the type of the variable.

PHP Data Types

PHP supports different types of data:





`String`: A sequence of characters.

Example:

```php
<?php
  $name = "John Doe";
  echo $name; // Output: John Doe
?>
```

`Integer`: Whole numbers (positive or negative).

Example:

```php
<?php
  $age = 25;
  echo $age; // Output: 25
?>
```

`Float`: Decimal numbers.

Example:

```php
<?php
  $price = 19.99;
  echo $price; // Output: 19.99
?>
```

`Boolean`: True or false values.

Example:

```php
<?php
  $isAvailable = true;
  echo $isAvailable; // Output: 1 (true)
?>
```

`Array`: A collection of values.

Example:

```php
<?php
  $colors = array("Red", "Green", "Blue");
  echo $colors[0]; // Output: Red
?>
```

Constants

Constants are like variables, but once defined, they cannot be changed. They are defined using the define() function.

```php
<?php
  define("SITE_NAME", "My Website");
  echo SITE_NAME; // Output: My Website
?>
```

|Task|---------->

```
Create a PHP script that defines variables for a person's name, age, and favorite color. Display these values on the web page.

Create constants for the website's title and display it in the page header.
```

PHP Operators and Control Structures

Operators in PHP

PHP supports several types of operators used to perform operations on variables and values.

1. Arithmetic Operators

Used for performing arithmetic operations like addition, subtraction, multiplication, etc.

Operator	Description	Example
+	Addition	$x + $y
-	Subtraction	$x - $y
*	Multiplication	$x * $y
/	Division	$x / $y
%	Modulus (remainder)	$x % $y

Example: Arithmetic Operators

```php
<?php
  $x = 10;
  $y = 5;
  echo $x + $y;  // Output: 15
  echo $x * $y;  // Output: 50
?>
```

2. Comparison Operators

Used for comparing values and returning a boolean result. Used to compare two data.

Operator	Description	Example
==	Equal to	$x == $y
!=	Not equal to	$x != $y
>	Greater than	$x > $y
<	Less than	$x < $y
>=	Greater than or equal to	$x >= $y
<=	Less than or equal to	$x <= $y
===	Identical to	$x === $y
!==	Not identical to	$x !== $y
<=>	Spaceship operator	$x <=> $y

Example: Comparison Operators

```php
<?php
$x = 10;
$y = 5;
echo $x == $y;  // Output: false
?>
```

3. Logical Operators

Used for combining multiple conditions to determine the final result.

Operator	Description	Example
!	Logical not	!$x
&&	Logical and	$x && $y
||	Logical or	$x || $y

Example: Logical Operators

```php
<?php
$x = true;
$y = false;
echo !$x;  // Output: false
echo $x && $y;  // Output: false
?>
```

4. Assignment Operators

Used to assign a value to a variable.

Operator	Description	Example

=	Simple assignment	$x = $y
+=	Addition assignment	$x += $y
-=	Subtraction assignment	$x -= $y
*=	Multiplication assignment	$x *= $y
/=	Division assignment	$x /= $y
%=	Modulus assignment	$x %= $y
**=	Exponentiation assignment	$x **= $y
<<=	Left shift assignment	$x <<= $y
>>=	Right shift assignment	$x >>= $y
&=	Bitwise AND assignment	$x &= $y
^=	Bitwise XOR assignment	$x ^= $y
|=	Bitwise OR assignment	$x |= $y

Example: Assignment Operators

```php
<?php
$x = 10;
$y = 5;
echo $x += $y;  // Output: 15
?>
```

5. Bitwise Operators

Used for performing operations on binary numbers.

Operator	Description	Example
&	Bitwise AND	$x & $y
|	Bitwise OR	$x | $y
^	Bitwise XOR	$x ^ $y
~	Bitwise NOT	~$x
<<	Left shift	$x << $y
>>	Right shift	$x >> $y
>>>	Zero-fill right shift	$x >>> $y

Example: Bitwise Operators

```php
<?php
$x = 5;  // Binary: 0101
$y = 3;  // Binary: 0011
echo $x & $y;  // Output: 1 (Binary: 0  001)
?>
```

Control Structures in PHP

Control structures allow you to control the flow of your program. PHP supports various types of control structures such as if, else, switch, loops, etc.

1. Conditional Statements

Used to execute code only if a certain condition is met.

if Statement

```php
<?php
  $age = 18;
  if ($age >= 18) {
    echo "You are eligible to vote.";
  }
?>
```

if-else Statement

```php
<?php
  $age = 17;
  if ($age >= 18) {
    echo "You are eligible to vote.";
  } else {
    echo "You are not eligible to vote.";
  }
?>
```

switch Statement

Used when you need to check multiple conditions.

```php
<?php
  $day = "Monday";
  switch ($day) {
    case "Monday":
      echo "It's Monday!";
      break;
    case "Tuesday":
      echo "It's Tuesday!";
      break;
    default:
      echo "It's some other day!";
  }
?>
```

2. Loops

Loops are used to repeat a block of code a specified number of times.

for Loop

```php
<?php
  for ($i = 0; $i < 5; $i++) {
    echo $i . "<br>";
  }
?>
```

while Loop

```php
<?php
  $i = 0;
  while ($i < 5) {
    echo $i . "<br>";
    $i++;
  }
?>
```

do-while Loop

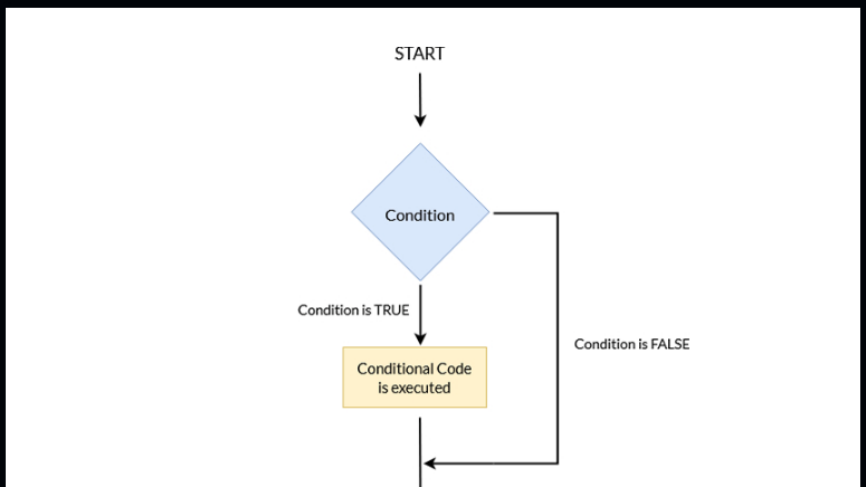
```php
<?php
$i = 0;
do {
  echo $i . "<br>";
  $i++;
  } while ($i < 5);
?>
```

foreach Loop (for arrays)

```php
<?php
  $colors = array("Red", "Green", "Blue");
  foreach ($colors as $color) {
    echo $color . "<br>";
  }
?>
```

foreach Loop (for associate arrays)

```php
<?php
$person = array("name" => "John", "age" => 30, "mobileno" => 4784548456);
foreach ($person as $key => $value) {
  echo $key . ": " . $value . "<br>";
}
?>
```



|Task|---------->

```
Create a PHP script that checks whether a number is positive, negative, or zero using conditional statements, and displays the
```

PHP Functions

What are Functions?

Functions are blocks of code that can be repeatedly called to perform specific tasks. They help avoid redundancy and make code easier to maintain.

Defining and Calling a Function

A function is defined using the `function` keyword followed by the function name and parentheses `()`.

Example: Defining and Calling a Function

```php
<?php
  function greet() {
    echo "Hello, World!";
  }

  greet(); // Output: Hello, World!
?>
```

Function Parameters

Functions can accept parameters (arguments) to perform tasks based on input values.

Example: Function with Parameters

```php
<?php
// Formal argument call when receive data

// Actual argument call when data pass
  function greet($name) {
    echo "Hello, " . $name;
  }

  greet("John"); // Output: Hello, John
?>
```

Function Return Values

Functions can return a value to the caller using the return statement.

Example: Returning a Value from a Function

```php
<?php
  function add($x, $y) {
    return $x + $y;
  }

  $sum = add(5, 10);
  echo $sum; // Output: 15
?>
```

Types of functions in PHP

Built-in Functions

PHP has a wide range of built-in functions that can be used to perform various tasks, such as

- String manipulation functions (e.g., `strlen()`, `strpos()`, `substr()
- Array manipulation functions (e.g., `array_push()`, `array_pop()`, `
- Math functions (e.g., `sin()`, `cos()`, `tan()`,
- Date and time functions (e.g., `date()`, `time()`, `
- And many more.

User-Defined Functions

User-defined functions are created using the `function` keyword and can be used to perform custom tasks.

Ways to define functions

1. Take something , Return something

```php
<?php
function add($x, $y) {
    return $x + $y;
    }

$res = add(45,65);
echo $res; // Output: 110
?>
```

2. Take something , Return nothing

```php
<?php
function greet($name) {
    echo "Hello, $name";
    }
    greet("John");
    ?>
```

3. Take nothing , Return something

```php
<?php
function get_date() {
    return date("Y-m-d");
}
$res = get_date();
echo $res; // Output: Current Date
?>
```

4. Take nothing , Return nothing

```php
<?php
function say_hello() {
    echo "Hello";
}
say_hello(); // Output: Hello
?>
```

Default Parameter Values

If a function is called without passing an argument, the default value will be used.

Example: Default Parameters

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

  greet();        // Output: Hello, Guest
  greet("John");  // Output: Hello, John
?>
```

Passing Arguments by Reference

PHP allows you to pass arguments by reference, meaning the actual value of the variable is passed and not just a copy.

Example: Passing by Reference

```php
<?php
  function addFive(&$num) {
    $num += 5;
  }

  $value = 10;
  addFive($value);
  echo $value; // Output: 15
?>
```

Functions with Multiple Return Values

PHP allows you to return multiple values from a function using an array or an object.

```php
<?php
function getValues() {
    return array(1, 2, 3);
    }
$x,$y,$z = getValues();
echo $x; // Output: 1
echo $y; // Output: 2
echo $z; // Output: 3
?>
```

Functions with Variable Number of Arguments

PHP allows you to define functions that can accept a variable number of arguments using the `func_get_args` function.

```php
<?php
function sum() {
    $sum = 0;
    $args = func_get_args();
    foreach ($args as $arg) {
        $sum += $arg;
        }
        return $sum;
}
echo sum(1, 2, 3, 4, 5); // Output:15
?>
```

Variable Scope

There are different scopes for variables in PHP:

1. `Global Scope` : Variables declared outside of any function.
2. `Local Scope` : Variables declared inside a function.

Example: Local and Global Variables

```php
<?php
  $x = 10; // Global scope

  function test() {
    $y = 5; // Local scope
    echo $y; // Output: 5
  }

  test();
  echo $x; // Output: 10

$z = 10;  // global variable

// for access global variable inside local scope one way
function my_function() {
    global $z;  // access global variable $z in local scope
    echo $z;  // Output: 10
}

my_function();


// for access global variable inside local scope another way
$g = 10;  // global variable

function my_function() {
    echo $GLOBALS['g'];  // Output: 10
}

my_function();

?>
```

|Task|---------->

```
Write a function that accepts an array of numbers and returns the sum of all numbers.
Write a function that accepts a string and returns the string in reverse.
```

PHP Arrays

What are Arrays?

An array is a data structure that allows you to store multiple values in a single variable. Arrays are useful for handling lists of data, such as items in a shopping cart or user information.

Types of Arrays in PHP

1. **Indexed Arrays:** Arrays with a numeric index.
2. **Associative Arrays:** Arrays where keys are named.
3. **Multidimensional Arrays:** Arrays containing one or more arrays.

---

1. Indexed Arrays

Indexed arrays use numeric keys to access elements.

Example: Indexed Array

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

```php
  // another way to declare array
  $arr = [45,6,8,8,8];
  echo $fruits[0]; // Output: Apple
?>
```

Example: Using for Loop with Indexed Arrays

```php
<?php
  $colors = array("Red", "Green", "Blue");
  for ($i = 0; $i < count($colors); $i++) {
    echo $colors[$i] . "<br>";
  }
?>
```

2. Associative Arrays

Associative arrays use named keys to access values.

Example: Associative Array

```php
<?php
  $person = array("Name" => "John", "Age" => 25, "City" => "New York");
  echo $person["Name"]; // Output: John
?>
```

Example: Looping through Associative Arrays with foreach

```php
<?php
  $person = array("Name" => "John", "Age" => 25, "City" => "New York");
  foreach ($person as $key => $value) {
    echo $key . ": " . $value . "<br>";
  }
?>
```

3. Multidimensional Arrays

Multidimensional arrays contain arrays within arrays.

Example: Multidimensional Array

```php
<?php
  $people = array(
    array("John", 25, "New York"),
    array("Jane", 28, "Los Angeles"),
    array("Tom", 22, "Chicago")
  );

  echo $people[0][0]; // Output: John
?>
```

Example: Looping through Multidimensional Arrays

```php
<?php
  $people = array(
    array("John", 25, "New York"),
    array("Jane", 28, "Los Angeles"),
    array("Tom", 22, "Chicago")
  );

  for ($i = 0; $i < count($people); $i++) {
    for ($j = 0; $j < count($people[$i]); $j++) {
      echo $people[$i][$j] . " ";
    }
    echo "<br>";
  }
?>
```

Array Functions in PHP

PHP provides several built-in functions to work with arrays:

Function	Description	Example
`array()`	Creates a new array	`array(1, 2, 3)`
`count()`	Returns the number of elements in an array	`count($array)`
`array_push()`	Adds one or more elements to an array	`array_push($array, "Element")`
`array_merge()`	Merges two or more arrays	`array_merge($array1, $array2)`
`array_keys()`	Returns all the keys of an array	`array_keys($array)`
`array_values()`	Returns all the values of an array	`array_values($array)`

|Task|---------->

```
Create an indexed array of car brands and display the first and last brands.
Create an associative array for a person with their name, age, and city. Display all the details using a loop.
Create a multidimensional array of students and their grades. Loop through and display each student's name and grade.
```

Working with Forms in PHP

Introduction to Forms

Forms are used to collect user input and send it to the server for processing. PHP is often used to handle form data sent via the HTTP POST or GET method.

HTML Form Example

The following is a simple HTML form that collects a user's name and email:

```
<form action="submit.php" method="POST">
  Name: <input type="text" name="name"><br>
  Email: <input type="email" name="email"><br>
  <input type="submit" value="Submit">
</form>
```

- The action attribute specifies the file where form data will be sent (in this case, submit.php).
- The method attribute defines the HTTP method to use when sending form data. Typically, you use POST or GET.

Handling Form Data in PHP

$_POST and $_GET Superglobals

PHP provides two superglobal arrays to collect form data:

$_POST: Used to collect form data sent via the POST method.

$_GET: Used to collect form data sent via the GET method.

Example: Handling Form Data with $_POST or $_GET

```
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST['name'];
    $email = $_POST['email'];
    echo "Name: " . $name . "<br>";
    echo "Email: " . $email . "<br>";
  }
  else if($_SERVER['REQUEST_METHOD'] == 'GET'){
    $name = $_GET['name'];
    $email = $_GET['email'];
    echo "Name: " . $name . "<br>";
    echo "Email: " . $email . "<br>";
  }
?>
```

Form Validation

It's important to validate form data to ensure that the input is correct and secure. PHP can perform both server-side validation (required for security) and client-side validation (optional but improves user experience).

Example: Validating a Required Field

```
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
      echo "Name is required";
    } else {
      echo "Name: " . $_POST["name"];
    }
  }
?>
```

Sanitizing and Validating Form Data

PHP provides functions to sanitize and validate form inputs to prevent security issues like XSS (Cross-Site Scripting) and SQL Injection.

Example: Sanitizing and Validating an Email Address

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

  // Remove illegal characters from email
  $email = filter_var($email, FILTER_SANITIZE_EMAIL);

  // Validate email
  if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Valid email address";
```

```
  } else {
    echo "Invalid email address";
  }
?>
```

|Task|---------->

```
Create a form that collects a user's name, email, and message. On form submission, display the entered details.
Add validation to ensure that the name and email fields are not empty, and that the email is a valid address.
```

Sessions and Cookies in PHP

What are Sessions?

Sessions in PHP allow you to store information across multiple pages. Sessions store data on the server, making them more secure than cookies for sensitive data.

Starting a Session

To start a session in PHP, use the `session_start()` function. This function must be called at the beginning of every script that uses sessions.

Example: Starting a Session and Storing Data

```
<?php
  session_start();  // Start the session
  $_SESSION['username'] = "JohnDoe";  // Store session data
  echo "Session started and data stored.";
?>
```

Accessing Session Data

Session data can be accessed on any page once the session is started.

Example: Accessing Session Data

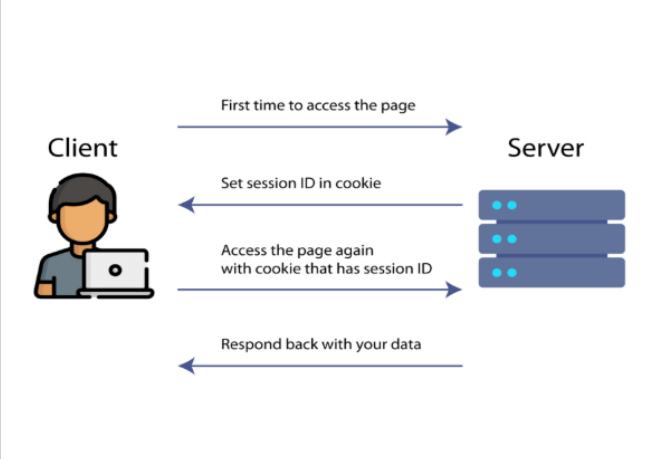
```
<?php
  session_start();  // Start the session
  echo "Username: " . $_SESSION['username'];  // Output: Username: JohnDoe
?>
```

Destroying a Session

To remove all session data and destroy the session, use the session_destroy() function.

Example: Destroying a Session

```
<?php
  session_start();
  // unset last session variable
  unset($_SESSION['username']);
  // destroy session data
  session_destroy();  // Destroy the session
  echo "Session destroyed.";
?>
```



What are Cookies?

Cookies are small text files stored on the client (user's browser). They are useful for tracking user behavior and storing non-sensitive data across pages.

Setting a Cookie

Cookies are set using the setcookie() function, which must be called before any HTML output.

Example: Setting a Cookie

```php
<?php
  setcookie("username", "JohnDoe", time() + (86400 * 30), "/");  // Cookie expires in 30 days
  echo "Cookie set.";
?>
```

Accessing a Cookie

Once a cookie is set, it can be accessed using the $_COOKIE superglobal.

Example: Accessing a Cookie

```php
<?php
  if (isset($_COOKIE['username'])) {
    echo "Username: " . $_COOKIE['username'];  // Output: Username: JohnDoe
  }
?>
```

Deleting a Cookie

To delete a cookie, set its expiration time to a past date.

Example: Deleting a Cookie

```php
<?php
  setcookie("username", "", time() - 3600, "/");  // Expire the cookie
  echo "Cookie deleted.";

// Explaination of setcookie function
// setcookie(name, value, expire, path)
// name: name of the cookie
// value: value of the cookie
// expire: expiration time of the cookie
// path: path of the cookie
?>
```

|Task|---------->

```
Create a login page that stores the username in a session after successful login. Create a separate page that welcomes the user
Set a cookie to store the user's preferred language for a website. Display a message in the selected language when the user rev
```

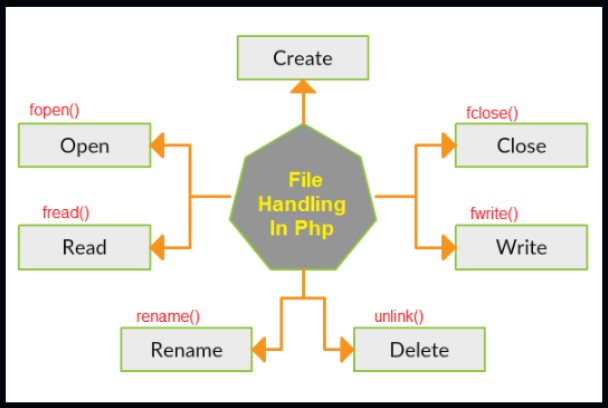
File Handling in PHP

Introduction to File Handling

PHP provides a variety of functions for handling files, such as reading, writing, and creating files. File handling is essential for tasks like storing logs, exporting data, or managing uploaded files.

Basic File Operations

1. **Opening a File:** Use the `fopen()` function to open a file.
2. **Reading from a File:** Use functions like `fread()`, `fgets()`, or `file_get_contents()` to read the contents of a file.
3. **Writing to a File:** Use `fwrite()` to write data to a file.
4. **Closing a File:** After completing operations, close the file using `fclose()`.



1. Opening and Closing a File

You can open a file using the `fopen()` function, which requires two arguments: the file path and the mode (e.g., read, write).

Example: Opening and Closing a File

```php
<?php
  $file = fopen("example.txt", "r");  // Open file in read mode
  fclose($file);                      // Close the file
?>
```

File Modes

Mode	Description
`r`	Open for reading only. The file pointer is positioned at the beginning of the file.
`w`	Open for writing only. If the file does not exist, it is created
`a`	Open for appending. If the file does not exist, it is created.
`r+`	Open for reading and writing. The file pointer is positioned at the beginning
`w+`	Open for reading and writing.
`a+`	Open for reading and appending.
`x`	Create and open for writing. If the file already exists, the operation fails

2. Reading from a File

There are multiple ways to read from a file in PHP:

fread(): Reads a specific number of bytes from a file.

fgets(): Reads a line from a file.

file_get_contents(): Reads the entire file into a string.

Example: Reading from a File

```php
<?php
  $file = fopen("example.txt", "r");
  while(!feof($file)) {
    $line = fgets($file);  // Read a line
    echo $line . "<br>";
  }
  fclose($file);

  // fread() use
  $content = fread($file, filesize("example.txt"));
  echo $content;

  // file_get_contents() use in php
  $content = file_get_contents("example.txt");
  echo $content;


?>
```

3. Writing to a File

To write data to a file, use the fwrite() function. Make sure the file is opened in write or append mode.

Example: Writing to a File

```php
<?php
  $file = fopen("example.txt", "w");
  fwrite($file, "Hello, World!\n");  // Write to file
  fclose($file);
?>
```

4. Deleting a File

To delete a file, use the unlink() function.

Example: Deleting a File

```php
<?php
  if (file_exists("example.txt")) {
    unlink("example.txt");  // Delete the file
    echo "File deleted.";
  } else {
    echo "File not found.";
  }
?>
```

5. Types of File

1. Binary File

2. Text File

```php
<?php
// Binary File
$file = fopen("example.bin", "rb");
$contents = fread($file, filesize("example.bin"));
fclose($file);
// Text File
$file = fopen("example.txt", "r");
$contents = fread($file, filesize("example.txt"));
fclose($file);
?>
//   CRUD Operation on binary file
```

6. File Operations

1. Copying a File
2. Renaming a File
3. Moving a File
4. Checking if a File Exists

```php
<?php
// Copying a File
$file = fopen("example.txt", "r");
$contents = fread($file, filesize("example.txt"));
fclose($file);
$file = fopen("example_copy.txt", "w");
fwrite($file, $contents);
fclose($file);

// Renaming a File
$file = fopen("example.txt", "r");
$contents = fread($file, filesize("example.txt"));
fclose($file);
$file = fopen("example.txt", "w");
rename("example.txt", "example_new.txt");

// Moving a File
$file = fopen("example.txt", "r");
$contents = fread($file, filesize("example.txt"));
fclose($file);
$file = fopen("example.txt", "w");
rename("example.txt", "/path/to/new/location/example.txt");

// Checking if a File Exists
$file = fopen("example.txt", "r");
if (file_exists("example.txt")) {
  echo "File exists";
  } else {
    echo "File does not exist";
  }
?>
```

7. File Uploading

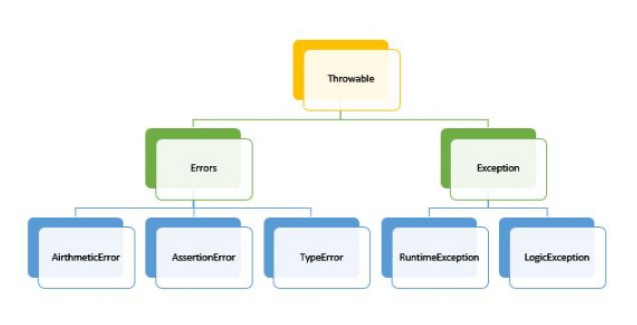
```php
<?php
// File Uploading
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

// Check if image file is a actual image or fake image

if (isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if ($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}

// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}

// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}

// Allow certain file formats
if ($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg" && $imageFileType != "gif") {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}

// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
```

```
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file " . basename($_FILES["fileToUpload"]["name"]) . " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```

`|Task|---------->`

```
Create a PHP script that writes user input to a text file. Display the contents of the file on a different page.
Create a log file system where errors are written to a log file every time they occur. Read and display the log content on a wel
```

Error Handling in PHP



What is Error Handling?

Error handling is the process of catching and managing errors in your code to prevent the application from crashing or displaying undesirable messages. PHP provides built-in functions for error handling and debugging.

Types of Errors in PHP

1. **Parse Errors:** Occur when the PHP interpreter encounters a syntax error.
2. **Fatal Errors:** Occur when PHP cannot execute a script (e.g., calling an undefined function).
3. **Warning Errors:** Non-fatal errors that do not stop script execution.
4. **Notice Errors:** Informal warnings about code that may not work as expected.

Example: Triggering an Error

```php
<?php
  echo $undefined_variable;  // Notice error: Variable not defined
?>
```

Custom Error Handling

You can define custom error handling using the set_error_handler() function. This allows you to specify how errors should be handled or logged.

Example: Custom Error Handler

```php
<?php
  // Custom error handler function
  function customErrorHandler($errno, $errstr, $errfile, $errline) {
    echo "Error [$errno]: $errstr in $errfile on line $errline";
  }

  set_error_handler("customErrorHandler");

  echo $undefined_variable;  // Triggers custom error handler
?>
```

Logging Errors

PHP can log errors to a file using the error_log() function. This is useful for tracking errors in production environments without displaying them to the user.

Example: Logging an Error

```php
<?php
  $error_message = "Error occurred on " . date("Y-m-d H:i:s");
  error_log($error_message, 3, "errors.log");  // Log error to a file
  // explain working of error_log function
  // error_log function takes three parameters: message, level, and destination
```

```
  // message is the error message to be logged
  // level is the level of the error (0 = log, 1 = system log,
  // 2 = user log)
  // destination is the file where the error will be logged
?>
```

Displaying Errors

To display errors for debugging purposes, enable error reporting using the error_reporting() function.

Example: Displaying All Errors

```php
<?php
  ini_set("display_errors", 1);  // Enable error display
  error_reporting(E_ALL);        // Report all errors
  //ini_set parameters explain
  // display_errors: 1 to display errors, 0 to hide them
  // error_reporting: E_ALL to report all errors, E_ERROR to report only fatal errors

?>
```

Exception Handling in PHP

In addition to traditional error handling, PHP provides exception handling using try, catch, and throw statements. Exceptions are used to handle errors in a more controlled way.

Methods of Exception Handling in php

Example: Handling Exceptions

```php
<?php
  function divide($a, $b) {
    if ($b == 0) {
      throw new Exception("Division by zero.");
    }
    return $a / $b;
  }

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

try-catch-finally

```php
<?php
  try {
    // code that might throw an exception
    throw new Exception('An error occurred');
} catch (Exception $e) {
    // code that will execute if an exception is thrown
    echo 'Caught exception: ' . $e->getMessage();
} finally {
    // code that will execute regardless of whether an exception is thrown or not
    echo 'This code will always execute';
}
?>
```

Exception Handling using custom exception handler

```php
<?php
function customExceptionHandler($e) {
    // custom exception handling code
    echo 'Caught exception: ' . $e->getMessage();
}
// The set_exception_handler function is used to set a custom exception handler function. This function will be called whenever
set_exception_handler('customExceptionHandler');

// code that might throw an exception
throw new Exception('An error occurred');

?>
```

|Task|---------->

```
Create a custom error handler that logs errors to a file instead of displaying them to the user.
Implement an exception handling system that handles division by zero errors and logs them in a separate log file.
```

DataBase Connection

Connecting php to a database using mysqli

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

CRUD operation using mysqli

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// SQL Query for Creating Table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
  echo "Table MyGuests created successfully";
} else {
  echo "Error creating table: " . $conn->error;
}

// Insert Data into Table
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('$firstname', '$lastname', '$email')";

if ($conn->query($sql) === TRUE) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

// Select Data from Table
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo "0 results";
}

// Update Data in Table
$firstname = "John";
```

```php
$lastname = "Doe";
$email = "johndoe@example.com";
$id = 1;

$sql = "UPDATE MyGuests SET firstname='$firstname', lastname='$lastname', email='$email' WHERE id=$id";

if ($conn->query($sql) === TRUE) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . $conn->error;
}

// Delete Data from Table
$id = 1;

$sql = "DELETE FROM MyGuests WHERE id=$id";

if ($conn->query($sql) === TRUE) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . $conn->error;
}

$conn->close();
?>
```

Executing Queries

You can execute queries using MySQLi's query() method and prepare statements using prepare().

Example: Executing a Simple Query

```php
<?php
$result = $mysqli->query('SELECT * FROM users');
$users = $result->fetch_all(MYSQLI_ASSOC);
print_r($users);
?>
```

Example: Using Prepared Statements

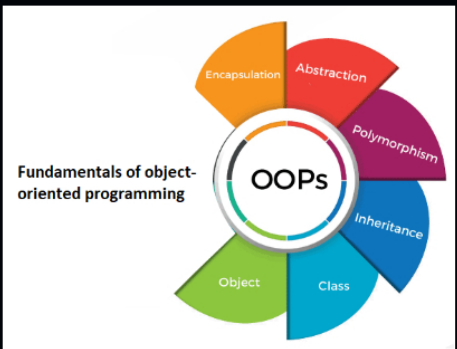
```php
<?php
$stmt = $mysqli->prepare('SELECT * FROM users WHERE id = ?');
$stmt->bind_param('i', $id);
$id = 1;
$stmt->execute();
$result = $stmt->get_result();
$user = $result->fetch_assoc();
print_r($user);
?>
```

|Task|---------->

```
Create a PHP script that connects to a MySQL database using both PDO and MySQLi.
Implement data retrieval using both PDO and MySQLi and display the results.
```

<==========ADVANC'E==========>

Object-Oriented Programming (OOP) in PHP



What is OOP?

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around objects, which can contain data and methods. In PHP, OOP provides a way to structure programs in a more modular, reusable, and scalable manner.

Key Concepts of OOP

1. Classes and Objects

- `Class` : A blueprint for creating objects. It defines properties (attributes) and methods (behaviors).
- `Object` : An instance of a class.

Example: Creating a Class and Object

```php
<?php
  class Car {
    public $brand;
    public $color;

    public function drive() {
      echo "The car is driving.";
    }
  }

  $myCar = new Car();
  $myCar->brand = "Toyota";
  $myCar->color = "Red";
  echo $myCar->brand;  // Output: Toyota
  $myCar->drive();     // Output: The car is driving.
?>
```

2. Properties and Methods

`Properties` : Variables inside a class (also called attributes).

`Methods` : Functions inside a class.

Example: Adding Properties and Methods

```php
<?php
  class Person {
    public $name;
    public $age;

    public function introduce() {
      echo "Hi, my name is " . $this->name . " and I am " . $this->age . " years old.";
    }
  }

  $john = new Person();
  $john->name = "John";
  $john->age = 30;
  $john->introduce();  // Output: Hi, my name is John and I am 30 years old.
?>
```

3. Constructor and Destructor

`Constructor` : A special method that is automatically called when an object is created.

`Destructor` : A special method that is called when an object is destroyed (when the script ends or the object is no longer needed).

Example: Constructor and Destructor

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

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

    // Destructor
    public function __destruct() {
      echo "The animal " . $this->name . " is destroyed.";
    }
  }

  $dog = new Animal("Dog");
  echo $dog->name;  // Output: Dog
  // Destructor is automatically called at the end
?>
```

4. Inheritance

Inheritance allows a class to use properties and methods of another class.

Example: Inheritance in PHP

```php
<?php
  class Vehicle {
    public $brand;

    public function start() {
      echo "Vehicle started.";
    }
  }

  class Car extends Vehicle {
    public function drive() {
      echo "The car is driving.";
    }
  }
```

```php
  $myCar = new Car();
  $myCar->brand = "Honda";
  $myCar->start();   // Output: Vehicle started.
  $myCar->drive();   // Output: The car is driving.
?>
```

5. Access Modifiers

Access modifiers define the visibility of properties and methods. PHP has three access modifiers:

public: Accessible from anywhere. private: Accessible only within the class. protected: Accessible within the class and by derived classes.

Example: Using Access Modifiers

```php
<?php
  class House {
    public $color = "White";     // Public
    private $owner = "John";     // Private
    protected $price = 50000;    // Protected

    public function showOwner() {
      echo $this->owner;
    }
  }

  $myHouse = new House();
  echo $myHouse->color;    // Output: White
  // echo $myHouse->owner; // Error: Cannot access private property
  $myHouse->showOwner();   // Output: John
?>
```

6. Encapsulation

Encapsulation is the concept of bundling data and methods that operate on that data within a single unit

Example: Encapsulation

```php
<?php
class BankAccount {
    private $balance = 0;
    public function deposit($amount) {
        $this->balance += $amount;
    }
    public function getBalance() {
        return $this->balance;
    }
}
$account = new BankAccount();
$account->deposit(100);
echo $account->getBalance(); // Output: 100
?>
```

7. Polymorphism

Polymorphism is the ability of an object to take on multiple forms

Example: Polymorphism

```php
<?php
class Animal {
    public function sound() {
        echo "The animal makes a sound.\n";
        }
}
class Dog extends Animal {
    public function sound() {
        echo "The dog barks.\n";
        }
}
class Cat extends Animal {
    public function sound() {
        echo "The cat meows.\n";
        }
}
$animals = array(new Dog(), new Cat());
foreach ($animals as $animal) {
    $animal->sound();
    }
?>
```

8. Abstract Class & Methods

`Abstract class` is a class that cannot be instantiated and is used to provide a common interface for its subclasses

`Abstract Method` is a method that is declared without an implementation

Example: Abstract Class & Methods

```php
<?php
abstract class Shape {
    abstract public function area();
    abstract public function perimeter();
    }
```

```
class Circle extends Shape {
        private $radius;
        public function __construct($radius) {
            $this->radius = $radius;
            }
            public function area() {
                return M_PI * pow($this->radius, 2);
            }
            public function perimeter() {
                return 2 * M_PI * $this->radius;
            }
}

class Rectangle extends Shape {
                private $length;
                private $width;
                public function __construct($length, $width) {
                    $this->length = $length;
                    $this->width = $width;
                    }
                    public function area() {
                        return $this->length * $this->width;
                    }
                    public function perimeter() {
                        return 2 * ($this->length + $this->width);
                    }
}
?>
```

9. NameSpace

Namespaces are used to group related classes, interfaces, functions, and constants together to avoid naming conflicts

Example: NameSpace

```
<?php
namespace MyNamespace;
class MyClass {
    public function myMethod() {
        echo "Hello, World!";
    }
}
?>
// use of MyNamespace
$myObject = new MyNamespace\MyClass();
$myObject->myMethod();

 // use of MyNamespace in other file
require_once 'MyNamespace.php';
```

Example: use of NameSpace in another file

```
<?php

// Import the namespace
use MyNamespace\MyClass;

// Create an object of the MyClass class
$obj = new MyClass();

// Call the myMethod method
$obj->myMethod();
?>
```

Another way to use name space

```
<?php

// Create an object of the MyClass class
$obj = new MyNamespace\MyClass();

// Call the myMethod method
$obj->myMethod();
?>
```

10. Traits

Traits are reusable blocks of code that can be used in multiple classes to avoid code duplication

Example: Traits

```
<?php
trait MyTrait {
    public function myMethod() {
        echo "Hello, World!";
    }
}
class MyClass {
    use MyTrait;
    public function myOtherMethod() {
        echo "This is another method";
        }
}
```

```
>>
```

|Task|---------->

```
Create a class Animal with properties name and type, and methods introduce() and sound(). Create an object and demonstrate call:
Create a class Bird that extends the Animal class and adds a method fly(). Demonstrate inheritance by creating a bird object.
```