UNIT - 1

Introduction to PHP

PHP stands for Hypertext Preprocessor. It's a server-side scripting language used for web development. It’s an interpreted language, there is no need for compilation.

Features of PHP

1. **Embedding in HTML:** PHP code can be embedded directly within HTML
2. **Object-Oriented Programming:** Build modular and reusable code.
3. **Database Interaction:** Connect to MySQL and others.
4. **Form Handling:** Process user input from web forms.
5. **Session Management:** Track user activity across visits.
6. **File Handling:** Manage files on the server.
7. **Dynamic Content:** Create pages that change based on user input.
8. **Easy to Learn:** Beginner-friendly syntax.
9. **Free and Open Source:** Free to use and modify.
10. **Versatility:** Handle various web development tasks.

Client-side Scripting Vs Server-Side Scripting

Feature	Server-Side Scripting	Client-Side Scripting
Location	Web server	User's web browser
Code visibility	Hidden from users	Visible to users
Primary function	Data processing, database interaction, dynamic content	User interaction, dynamic visuals, basic form validation
Security	More secure due to hidden code	Less secure due to exposed code
Examples (languages)	PHP, Python, Ruby, Java	JavaScript
Examples (tasks)	User login verification, order processing, data storage	Form validation, animations, responding to user clicks

PHP Syntax Essentials

- PHP code resides within special tags: <?php ... ?>
- PHP files use the .php extension.
- PHP and HTML can coexist within a file.
- PHP is case-sensitive. Statements must end with a semicolon (;).

Variables

- Variables store data, declared with a $ followed by the name (case-sensitive).

Constants

- Represent fixed values that shouldn't change during script execution.
- **Declaration:** Use the `define()` function or the `const` keyword

Key Differences (Constants Vs Variables) :

- **Mutability:** Variables can be changed, constants cannot.
- **Declaration:** Variables use $, constants don't.
- **Scope:** Variables can be local or global, constants are always global.

Data Types in PHP

1. Primitive Data Types (Scalar Types): Hold single values

- **Integer (int):** Whole numbers (e.g., age)
- **Float (float):** Decimals (e.g., pi)
- **Boolean (bool):** True or False (e.g., isRegistered)
- **String (string):** Text (e.g., name)
- **NULL:** No value assigned

2. Compound Data Types: Store collections

- **Array:** Ordered collections of values, accessed by numeric or string keys.
- **Object:** Encapsulate data and functions

3. Special:

- **Resource:** References to external resources (files, databases)
- **Callable:** Functions or methods to be called later

Operators in PHP

Operators Perform actions on values (operands) and give a result.

Types:

1. Arithmetic (+, -, *, /, %): Perform basic math.
2. Assignment (=, +=, -=, *=, /=, %): Assign values (= for basic, others combine operations with =).
3. Comparison (==, !=, <, >, <=, >=): Compare values, return true/false based on condition.
4. Logical (&&, ||, !): Combine true/false expressions (AND, OR, NOT).

- && : AND
- || : OR
- ! : NOT

5. Increment/Decrement (++ --): Increase or decrease a variable's value by 1 (pre/post versions).
6. String Operators (. .=): Perform string concatenation operations.
7. Array Operators (+, ==, !=, ===, !==): Work with arrays (collections of values).

- +: Union (merges arrays)
- ==, !=, ===, !==: Comparisons (mostly equality checks)

8. Conditional Operator (?:): Assign value based on a condition (shorthand if-else).

Expressions

Combinations of variables, values, operators, and function calls.
Evaluated to produce a single result.

Control Statements in PHP

Control statements dictate the flow of execution.
They allow you to make decisions and repeat code blocks based on conditions.

Types of Control Statements:

1. Conditional Statements:

- **if statement:** Executes a block of code if a condition is true.
- **if-else statement:** If the condition is true, do something; otherwise, do something else.
- **elseif statement:** Allows for multiple conditions to be checked.
- **switch statement:** Evaluates an expression against multiple possible values.

2. Looping Statements:

- **for loop:** Executes a block of code a predetermined number of times.
- **while loop:** Executes a block of code as long as a condition is true.
- **do-while loop:** Similar to while loop, but the code block executes at least once.
- **foreach loop:** Used specifically to iterate over arrays.
- **Nested Loops:** A loop inside another loop. The inner loop runs completely for each iteration of the outer loop.

3. Jump Statements: alter the normal flow of execution by jumping to a specific point in the code.

- **break statement:** Exits a loop prematurely.
- **continue statement:** Skips the current iteration of a loop and continues to the next.
- **goto statement:** Jumps to a labeled section of code. Not recommended in modern PHP.

UNIT - 2

PHP Arrays

Allows you to store a collection of elements under a single variable name. Elements are accessed using an **index** (numeric for ordered lists) or a **key** (associative for named elements).

Create Arrays in PHP:

1. Using the `array()` function
2. Using the short array syntax (`[]`)
3. Using the `compact()` function

Benefits: Organize related data efficiently. Simplify complex data structures.Improve code readability and maintainability.

Types of Arrays in PHP

1. Indexed / Numeric Arrays: Ordered collections of values. Accessed using numeric indexes.
2. Associative Arrays: Unordered collections of key-value pairs. Keys can be strings or numbers.
3. Multidimensional Arrays: Arrays that contain other arrays. Accessed using multiple indexes.

PHP Functions

Functions allow you to group a block of code that performs a specific task. This code can be reused throughout your program by simply calling the function.

Types of Functions:

1. Built-in Functions: Built-in functions are pre-written tools in a programming language.
2. User-Defined Functions: Functions created by users to perform a task.

Defining a Functions:
Use the `function` keyword followed by the function name and parentheses.

Calling Functions:
Use the function name followed by parentheses `()`.

Parameter passing / Arguments (Optional):
Pass any required arguments (data) within the parentheses, separated by commas

Return Values (Optional):
Functions can return a value using the `return` statement.

Creating Strings

1. Single quotes ('): `$message = 'Hello, world!';`
2. Double quotes ("): `$path = "C:\\Users\\Sandy\\Documents";`

Accessing Strings

Indexing: Access individual characters using zero-based indexing within square brackets.

String functions: PHP provides various functions for manipulating strings. (check below)

String Manipulation Functions

1. Length and Case:

- `strlen($string)`: Returns the number of characters in a string.
- `strtoupper($string)`: Converts all characters in a string to uppercase.
- `strtolower($string)`: Converts all characters in a string to lowercase.

2. Searching and Replacement:

- `strpos($string, $search, $start)`: Finds the first occurrence of a substring.
- `stripos($string, $search, $start)`: Case-insensitive version of `strpos()`
- `str_replace($search, $replace, $string)`: Replaces all occurrences of a substring with another substring.

3. Trimming and Splitting:

- `trim($string, $charlist)`: Removes a set of characters from the beginning and end.
- `ltrim($string, $charlist)`: Removes a set of characters from the beginning.
- `rtrim($string, $charlist)`: Removes whitespace from the end of a string.
- `explode($separator, $string, $limit)`: Converts a string into an array by splitting

4. Other Useful Functions:

- `substr($string, $start, $length)`: Extracts a portion of a string.
- `str_repeat($string, $multiplier)`: Repeats a string a specified number of times.
- `str_split($string, $length)`: Splits a string into an array of characters.

Formatting Strings

1. Using `sprintf()` function:
Creates a formatted string by inserting placeholders with corresponding values. Similar to C.

- **Syntax:** `sprintf(format_string, arg1, arg2, ...)`
- **Formatting Codes:** (Within placeholders `%`)
  - Specify how to format the corresponding argument.
- Common codes:
  - `%s`: String
  - `%d`: Signed decimal number (integer)
  - `%f`: Floating-point number
  - `%c`: Single character
  - `%x`: Hexadecimal number (lowercase)
  - `%X`: Hexadecimal number (uppercase)
- `printf()` function for direct output formatting (doesn't return a string like `sprintf()`).

2. String Interpolation: Embed variables directly within double-quoted strings.

Pattern Matching

Pattern matching involves searching for specific patterns or regular expressions within a String. PHP provides functions like preg_match() for pattern matching.

1. Regular Expressions (Regex): Powerful tool for string manipulation and pattern searching.

- Functions like `preg_match()`, `preg_match_all()`, `preg_replace()`, and `preg_split()` facilitate various operations.

UNIT - 3

Form Data Handling

PHP uses superglobals (Always Accessible Variables) to access form data.

The two main ones for forms are:
`$_GET`: Used for data sent through the URL with the GET method (less secure, limited data size).
`$_POST`: Used for data sent hidden within the request body with the POST method.

Submitting Forms:

- HTML forms specify the action using the `action` attribute in the `<form>` tag. This points to the PHP script that will handle the data.
- The `method` attribute defines how the data is sent (GET or POST).

Superglobal Variables

These variables are accessible throughout your script without the need for global declarations.

1. **$_GET:** Captures data passed through a URL query string.
2. **$_POST:** Captures data submitted through an HTML form using the `POST` method.
3. **$_REQUEST:** Combines data from both $_GET and $_POST, along with cookies (if enabled).

Cookies

A cookie is a small piece of data (usually text) that a web server stores on the user's computer. Cookies are essential for maintaining user state across different web pages on the same website.

Need for Cookies

1. **State Management:** Remember user actions or preferences
2. **Authentication:** Cookies can store login information.
3. **Personalization:** Tailored content or recommendations based on user preferences.
4. **Tracking:** Commonly used in web analytics to track user behavior.

Setting Up a Cookie

`setcookie()` Function: To create and send a cookie to the user's browser.

Arguments: The function takes several arguments:

- `name`: The name for the cookie (string). ,
- `value`: The data to store in the cookie (string).
- `expire` , `path` , `domain` , `secure` , `httponly` : All these are optional

Deleting a Cookie

Set Expired Cookie: Set the `expire` argument in `setcookie()` to a time in the past.

Session Management

Sessions enable you to store user-specific information
Unlike cookies (which reside on the client-side), session data is stored on the server-side, enhancing security.

Creating Session Variables

Initiating a Session: start a session using the `session_start()` function before creating.
Creating Session Variables: Use the superglobal `$_SESSION` array to store session variables.

- $_SESSION[“username”] = “mrsandy”;

Retrieving Session Variables: Access variables using their names within square brackets.

- $_SESSION[“username”];

Removes a specific session variable: `session_unset()`
End a Session: `session_destroy()`

Exception Handling

Exception handling is a robust mechanism in PHP for managing runtime errors gracefully, preventing your application from crashing and enhancing code readability.

1. Errors: Unexpected conditions that halt script execution.
2. Exceptions: Objects representing unexpected events. Provide information about the error.
3. Exception Handling:

- Uses `try...catch...finally` blocks.
- `try`: Code that might throw an exception.
- `catch`: Captures a specific exception type and defines how to handle it.
- `finally` (optional): Code that always executes, regardless of exceptions.

User Defined Exceptions

Custom exceptions created by extending the built-in `Exception` class. Allow you to handle specific errors in your application.

- Improved code readability and maintainability. Clearer error messages for developers.

Creating a Custom Exception Class:

- Extend the `Exception` class.
- Optionally define a constructor to set the error message.
- (Optional) Add custom methods specific to your exception type.

Throwing Exceptions: Use the `throw` statement with your custom exception object.

Catching Exceptions: Use a `try...catch` block to handle thrown exceptions. Catch specific exception types or use a generic `Exception` catch.

UNIT - 4

File Handling

File handling refers to a set of functions in PHP that allow you to interact with files on the server's disk. This includes operations like creating, reading, writing, appending, and deleting files.

- Uploading files (e.g., user avatars). Saving user data (e.g., login credentials)
- Reading configuration files. Creating log files

Key Functions:

- `fopen()`: Opens a file for specific operations.
- `fread()`: Reads a portion of a file into a variable.
- `fwrite()`: Writes data to a file.
- `fclose()`: Closes a file after operations are complete.
- Additional functions: `filesize()`, `fgets()`, `feof()`, `file_exists()`, `unlink()`

Opening a File

Use the `fopen()` function.
Takes two arguments:

- `$filename`: Path to the file (including filename).
- `$mode`: Opening mode (read, write, append etc.).
- Common modes:
    - `'r'`: Read only (throws error if file doesn't exist).
    - `'w'`: Write only (creates new file or overwrites existing).
    - `'a'`: Append only (creates new file or writes to the end).
    - `'r+'`: Read and write (existing file).
    - `'w+'`: Read and write (creates new file or overwrites).

Closing a File

- Use the `fclose()` function.
- Takes one argument: the file handle returned by `fopen()`.

File Manipulation in PHP

- `copy()`**: Copy Files (** Takes two arguments: source file path and destination file path.)
- `rename()`**: Renaming Files (** Takes two arguments: old file path and new file path.)
- `unlink()`**: Deleting Files (** Takes one argument: the file path to delete.)
- `fread()` : Reads a specific number of bytes from file.
- `fgets()` : Reads a single line from the file.
- `file_get_contents()` : Reads the entire file content into a string.
- `fwrite()`: Write data to the file.

Database Handling

Database handling in PHP refers to the capability of PHP scripts to interact with databases. This allows web applications to store, retrieve, update, and delete data dynamically.

How it works:

- **Connection:** PHP establishes a connection to the database server using MySQLi or PDO. These extensions provide functions for connecting, executing queries, and handling results.
- **Interaction:** PHP scripts use SQL statements to interact with the database.
- **CRUD operations:** The core functionalities involve CRUD operations on the database tables.

Connecting to MySQL Database

1. MySQLi:

- Define connection details (hostname, username, password, database name).
- Use `mysqli_connect()` to establish connection.

2. PDO (PHP Data Objects): General-purpose interface, works with multiple database types.

Performing Basic Database Operations (CRUD)

1. Insert (Create): Use the `INSERT` statement with the table name and column names.
2. Select (Read): Use the `SELECT` statement to retrieve data from the table.
3. Update (Modify): Use the `UPDATE` statement to modify existing data in the table.
4. Delete (Remove): Use the `DELETE` statement to remove data from the table.
Additional: Filter data with `WHERE` clause.

Query Handling in PHP

- Use `mysqli_query(connection, query_string)` to execute queries.
- Function returns `mysqli_result` object for SELECT queries, `TRUE` for others.

Handling Results:

- Use functions like `mysqli_num_rows($result)` to get the number of rows.
- Fetch results row by row using:
  - `mysqli_fetch_assoc($result)` (associative array)
  - `mysqli_fetch_array($result)` (numeric array or both)
  - `mysqli_fetch_all($result)` (all rows into an array)

Important Functions:

- `mysqli_free_result($result)`: Frees memory associated with the result set.
- `mysqli_close($connection)`: Closes the connection to the database.