

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.
- Unlike HTML, PHP code is executed on the server before sending the final HTML page to the user's browser.

Features of PHP

- **Embedding in HTML:** PHP code can be embedded directly within HTML files using special tags (`<?php ... ?>`).
- **Object-Oriented Programming:** PHP supports object-oriented programming for modular and reusable code.
- **Database Interaction:** Connects to databases like MySQL to store and retrieve data.
- **Form Handling:** Processes form data submitted by users on web pages.
- **Session Management:** Tracks user activity across multiple page visits.
- **File Handling:** Creates, reads, writes, and manages files on the server.
- **Dynamic Content:** PHP generates dynamic web pages, meaning content can change based on user input or other factors.
- **Easy to Learn:** PHP has a relatively simple syntax making it beginner-friendly.
- **Free and Open Source:** Anyone can download and use PHP for free.
- **Versatility:** PHP can handle tasks like form data, databases, user authentication, and more.

Client-side Scripting vs. Server-side Scripting

Client-side scripting and **server-side scripting** are two techniques used in web development, but they handle different tasks and run on different locations.

1. Client-side Scripting

- Runs on the user's web browser
- Focuses on user interface (UI) interactivity and dynamic behavior
- Examples:
 - JavaScript for adding animations, form validation, or interactive elements
 - HTML forms for user input

- Advantages:
 - Faster response times for users
 - Creates a more dynamic and engaging experience
- Disadvantages:
 - Limited functionality without server interaction
 - Code is visible to users, potentially exposing sensitive logic

2. Server-side Scripting

- Runs on the web server
- Handles tasks like data processing, database access, and generating dynamic content
- Examples:
 - PHP, Python, Ruby for processing user input, interacting with databases, and creating web pages
- Advantages:
 - More secure as code is hidden from users
 - Access to server resources and databases
- Disadvantages:
 - Can introduce delays if server processing is slow

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

File Extension: PHP code resides in files with the `.php` extension.

Embedding in HTML: PHP code can be embedded within HTML documents using special tags:

- Opening tag: `<?php`
- Closing tag: `?>`

Case Sensitivity: PHP is case-sensitive. Variable names like `$name` and `$Name` are considered different. However, keywords like `if`, `else`, and `for` are not case-sensitive.

Semicolons: Statements in PHP must end with a semicolon (;).

Comments:

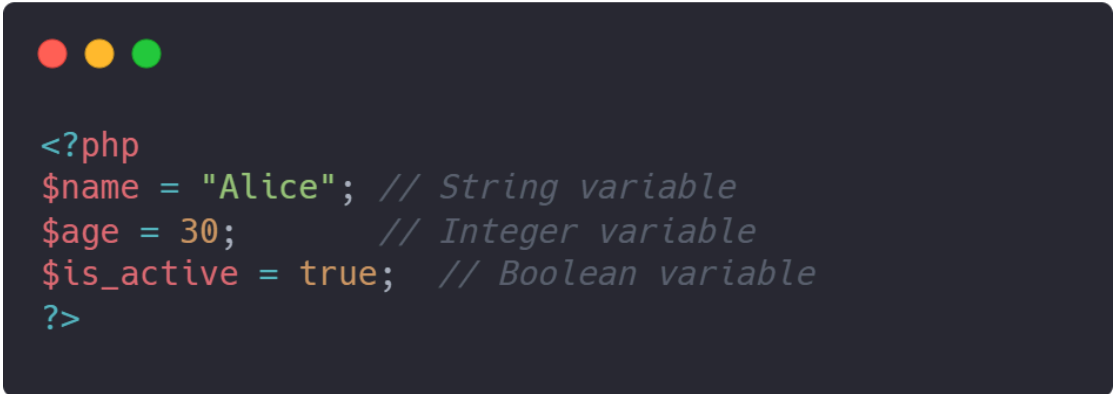
- Single-line comments: Use `//` followed by your comment.
- Multi-line comments: Use `/*` and `*/` to enclose your comment.

Variables:

- Variables store information and are declared with a dollar sign (\$) followed by the variable name.
- Variable names can contain letters, numbers, and underscores, but must start with a letter or underscore.

PHP Variables

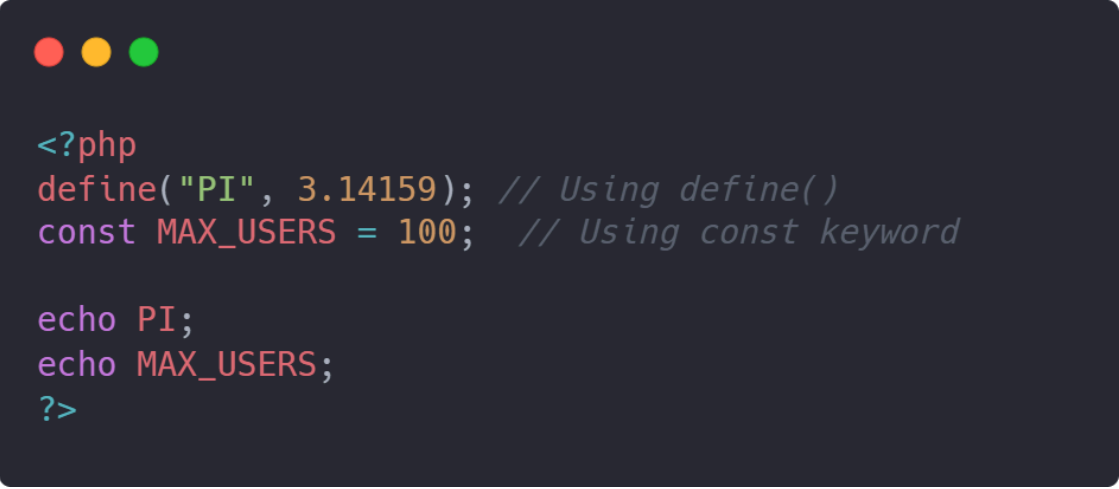
- **Purpose:** Store and manipulate data during program execution.
- **Declaration:** Use a dollar sign (\$) followed by a valid variable name (letters, underscores, no spaces).
- **Assignment:** Assign a value using the equal sign (=). Data type is determined automatically (loose typing).
- **Scope:** Can be local (within a function) or global (accessible throughout the script).
- **Example:**



```
<?php
$name = "Alice"; // String variable
$age = 30;       // Integer variable
$is_active = true; // Boolean variable
?>
```

PHP Constants

- **Purpose:** Represent fixed values that shouldn't change during script execution.
- **Declaration:** Use the `define()` function or the `const` keyword (PHP 5.6+).
- **Syntax:**
 - `define(name, value, case_insensitive);`
 - `const NAME = value;` (case-sensitive)
- **Value:** Numbers, strings, arrays (PHP 7+).
- **Scope:** Always global, accessible from anywhere in the script.
- **Example:**



```
<?php
define("PI", 3.14159); // Using define()
const MAX_USERS = 100; // Using const keyword

echo PI;
echo MAX_USERS;
?>
```

Key Differences:

- **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):** These basic building blocks hold single values.

- **Integer (`int`):** Whole numbers, positive, negative, or zero. (`$age = 25;`)
- **Float (`float`):** Decimal numbers. (`$pi = 3.14159;`)
- **Boolean (`bool`):** True or False values. (`$isRegistered = true;`)
- **String (`string`):** Sequences of characters, including text, numbers, and symbols.
 - Example: `$name = "Alice";`
- **NULL:** Represents the absence of a value. (`$unassignedVariable = null;`)

2. Compound Data Types: These structures can store collections of data.

- **Array (**array**):** Ordered collections of values, accessed by numeric or string keys.
 - Example: `$fruits = array("apple", "banana", "orange");`
- **Object (**object**):** Instances of classes, encapsulating data (properties) and behavior (methods).
 - Example: `$car = new Car();` (Assuming a `Car` class exists)

3. Special Data Types:

- **Resource (**resource**):** Handles references to external resources like files or database connections.
 - Created using functions like `fopen()` or `mysql_connect()` (deprecated in PHP 7+).
- **Callable (**callable**):** Represents functions or methods that can be invoked later.
 - Example: `$greet = function($name) { echo "Hello, $name!"; };`

Key Points:

- **PHP is loosely typed:** Variables don't have explicit data types declared beforehand. The type is determined by the value assigned.
- **Type juggling:** PHP can sometimes convert between types automatically.

Operators in PHP

- Perform operations on values (operands) and produce a result.
- Classified into different types based on their function.

1. Arithmetic Operators: (`+` , `-` , `*` , `/` , `%`)

- Perform basic mathematical calculations like addition, subtraction, multiplication, division, and modulus.
- `+` : Addition
- `-` : Subtraction
- `*` : Multiplication
- `/` : Division
- `%` : Modulus (remainder)
- Examples:

- `$sum = 10 + 5; // $sum will be 15`
- `$difference = 20 - 7; // $difference will be 13`

2. Assignment Operators: (`=` , `+=` , `-=` , `*=` , `/=` , `%=`)

- Assign values to variables.
- `=` is the basic assignment operator.
- `+=` , `-=` , `*=` , `/=` , `%=` : Combined assignment operators.
(add/subtract/multiply/divide/modulus and assign)
- Examples:
 - `$age = 30;`
 - `$message = "Hello World!";`

3. Comparison Operators: (`==` , `!=` , `<` , `>` , `<=` , `>=`)

- Compare values and return true or false based on the condition.
- `<` : Less than
- `>` : Greater than
- `==` : Equal to
- `!=` : Not equal to
- `<=` : Less than or equal to
- `>=` : Greater than or equal to
- Examples:
 - `$x == 10; // Checks if $x is equal to 10`
 - `$y > 25; // Checks if $y is greater than 25`

4. Logical Operators: (`&&` , `||` , `!`)

- Combine boolean expressions (true or false) using AND, OR, and NOT.
- `&&` : AND
- `||` : OR
- `!` : NOT
- Example:
 - `($age > 18) && ($has_valid_id == true); // Checks if age is above 18 and ID is valid`

5. Increment/Decrement Operators: (`++` , `--`)

- Increase or decrease a variable's value by 1.
- `++` for increment, `--` for decrement (pre-increment/decrement) or placed after the variable (post-increment/decrement).
- Examples:
 - `$count++; // Increment $count after using its current value`
 - `$likes = 10; $likes--; // Decrement $likes after assigning its current value to $likes`

6. String Operators: (`.` , `.=`)

- Perform operations on strings like concatenation.
- `.` : Concatenation
- `.=` : Concatenation assignment
- Examples:
 - `$name = "John"; $lastName = "Doe";`
 - `$fullname = $name . $lastname;`

7. Array Operators: (`+` , `==` , `!=` , `===` , `!==`)

- Used for working with arrays (collections of values).
- `+`: Union (merges arrays)
- `==`, `!=`, `===`, `!==`: Comparisons (mostly equality checks)
- Examples:
 - `$numbers = array(1, 2, 3); // Create an array`
 - `in_array(5, $numbers); // Check if 5 exists in $numbers (returns true/false)`

8. Conditional Operator (Ternary Operator): (`?:`)

- `?:` : Assigns a value based on a condition (shorthand if-else)
- Syntax:
 - `(condition) ? (expression_if_true) : (expression_if_false)`

Expressions

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

- `$total = $price * $quantity; // Expression to calculate total price`
- `if ($user_age >= 13) { // Expression in an if statement`

Control Statements in PHP


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

Types of Control Statements:

1. Conditional Statements:

Used to execute code based on whether a condition is true or false.


- **if statement:** Executes a block of code if a condition is true.



```
$age = 20;

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

- **if-else statement:** Executes one block of code if a condition is true, and another block if it's false.



```
$grade = 'B';

if ($grade === 'A') {
    echo "Excellent work!";
} else {
    echo "Keep practicing!";
}
```


- **elseif statement:** Allows for multiple conditions to be checked.

```
$day = 'Sunday';

if ($day === 'Saturday') {
    echo "It's the weekend!";
} elseif ($day === 'Sunday') {
    echo "Enjoy your Sunday!";
} else {
    echo "Back to work!";
}
```

- **switch statement:** Evaluates an expression against multiple possible values.

```
$choice = 'apple';

switch ($choice) {
    case 'apple':
        echo "You chose an apple.";
        break;
    case 'banana':
        echo "You chose a banana.";
        break;
    default:
        echo "Invalid choice.";
}
```


2. Looping Statements:

Used to execute a block of code repeatedly until a condition is met.


- **for loop:** Executes a block of code a predetermined number of times.

```
for ($i = 1; $i <= 5; $i++) {
    echo "Iteration number: $i <br>";
}
```

- **while loop:** Executes a block of code as long as a condition is true.

```
  
$x = 0;  
  
while ($x < 3) {  
    echo "Value of x: $x <br>";  
    $x++;  
}
```

- **do-while loop:** Similar to while loop, but the code block executes at least once.

```
  
$y = 10;  
  
do {  
    echo "Value of y: $y <br>";  
    $y--;  
} while ($y > 0);
```

- **foreach loop:** Used specifically to iterate over arrays.

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

- **Nested Loops**

Nested loops involve placing one loop (inner loop) entirely within the body of another loop (outer loop). The inner loop executes completely for each iteration of the outer loop.

- **Applications:** Nested loops are useful for tasks that require iterating through data structures with multiple dimensions, such as:
 - Multidimensional arrays
 - Matrices
 - Grid-based operations

```
for ($rows = 1; $rows <= 5; $rows++) { // controls the number of rows
    for ($cols = 1; $cols <= $rows; $cols++) { // Inner loop controls
        the number of asterisks in each row
        echo "*";
    }
    echo "<br>"; // Add a newline after each row
}
```

3. Jump Statements (Less Common)

Used to 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:** Not recommended in modern PHP, jumps to a labeled section of code.

```
<?php
$num = 5;

for ($i = 1; $i <= 10; $i++) {
    if ($i % 2 == 0) continue;
}

while ($num > 0) {
    if ($num == 7) break;
    $num--;
}

goto end;

echo " unreachable "; // unreachable due to goto

end:
?>
```