

PHP

PROGRAMMING

Elysium Academy Spark Notes

VERSION 2.4

01. PHP Basics

PHP is embedded within HTML using `<?php ... ?>` tags, making it very flexible for mixing PHP with other web technologies like HTML, CSS, and JavaScript.

Basic PHP Script

```
1. <?php
2.     echo "Hello, World!";
3. ?>
```

- **Key Points:**

- **PHP Code Blocks:** PHP code is wrapped within `<?php ?>` or `<?php echo ?>`.
- **Echo Statement:** echo is used to output text or variables to the browser.
- **Case Sensitivity:** PHP keywords are case-insensitive (echo, ECHO both work), but variables are case-sensitive.

- **Basic Syntax and Structure:**

- **Statements End with Semicolons:** Every PHP statement must end with a semicolon (;).
- **PHP Comments:** You can add comments with `//` (single line) or `/* ... */` (multi-line).

```
1. // This is a single-line comment
2. /* This is a
3.     multi-line comment */
```

02. PHP Data Types

PHP supports a wide range of data types, including primitive and complex types.

- **Primitive Data Types:**

Data Type	Description	Example
int	Integer numbers	\$age = 25;
float	Floating point numbers	\$price = 19.99;
string	Sequence of characters	\$name = "PHP";
bool	Boolean (True/False)	\$is_valid = true;
null	Represents a variable with no value	\$var = null;

• **Complex Data Types:**

Data Type	Description	Example
array	A collection of values	\$arr = [1, 2, 3];
object	An instance of a class	\$obj = new MyClass();
resource	Special type representing external resources	e.g., database connections

03. Variables and Constants

Variables in PHP are dynamic, meaning you don't need to declare their type explicitly. Variables are prefixed with \$ and can store any data type

• **Variable Declaration:**

```

1. $name = "John";           // String variable
2. $age = 30;                // Integer variable
3. $price = 19.99;          // Float variable
4. $is_valid = true;        // Boolean variable
  
```

• **Variable Scope:**

- **Local Variables:** Declared inside a function and used only within that function.
- **Global Variables:** Declared outside a function and can be accessed from anywhere in the script using the global keyword.
- **Static Variables:** Retain their value even after the function execution ends.

- **Constants:**

Constants in PHP are defined using the `define()` function or using `const`.

```
1. define("PI", 3.14159); // Using define()
2. const MAX_VALUE = 100; // Using const
```

- Constants are case-sensitive by default, but you can make them case-insensitive: `define("MAX_SPEED", 120, true);`

04. Operators in PHP

PHP offers a wide range of operators, including arithmetic, comparison, logical, assignment, and more.

- **Arithmetic Operators:**

Operator	Description	Example
+	Addition	$\$a + \b
-	Subtraction	$\$a - \b
*	Multiplication	$\$a * \b
/	Division	$\$a / \b
%	Modulus	$\$a \% \b

- **Comparison Operators:**

Operator	Description	Example
==	Equal to	$\$a == \b
!=	Not equal to	$\$a != \b
===	Identical (same value and type)	$\$a === \b
!==	Not identical	$\$a !== \b
>	Greater than	$\$a > \b
<	Less than	$\$a < \b

- **Logical Operators:**

Operator	Description	Example
&&	Logical AND	\$a && \$b
!	Logical NOT	!\$a

- **Assignment Operators:**

Operator	Description	Example
=	Assign value	\$a = \$b;
+=	Add and assign	\$a += \$b;
-=	Subtract and assign	\$a -= \$b;
*=	Multiply and assign	\$a *= \$b;
/=	Divide and assign	\$a /= \$b;

- **Increment/Decrement Operators:**

Operator	Description	Example
++\$a	Pre-increment (increment before use)	++\$a
\$a++	Post-increment (increment after use)	\$a++
--\$a	Pre-decrement (decrement before use)	--\$a
\$a--	Post-decrement (decrement after use)	\$a--

05. PHP Strings

PHP has a rich set of functions for manipulating strings.

- **String Declaration:**

```
1. $greeting = "Hello, World!";
```

- **Common String Functions:**

Function	Description	Example Usage
strlen()	Returns the length of the string	strlen(\$greeting)
strtoupper()	Converts string to upper-case	strtoupper(\$greeting)
strtolower()	Converts string to lower-case	strtolower(\$greeting)
strpos()	Finds the position of a substring	strpos(\$greeting, "World")
str_replace()	Replaces all occurrences of a substring	str_replace("World", "PHP", \$greeting)
substr()	Returns a substring	substr(\$greeting, 0, 5)
trim()	Removes whitespace from both ends	trim(" Hello ")

- **String Concatenation:**

- PHP uses the dot (.) operator for concatenation.

```

1. $firstName = "John";
2. $lastName = "Doe";
3. $fullName = $firstName . " " . $lastName;
4. echo $fullName; // Outputs: John Doe

```

06. Arrays

Arrays in PHP are used to store multiple values in a single variable. PHP supports both indexed and associative arrays.

- **Indexed Arrays:**

```
1. $fruits = array("Apple", "Banana", "Orange");
```

- **Associative Arrays:**

```
1. $ages = array("John" => 25, "Jane" => 30, "Mark" => 35);
```

- **Multidimensional Arrays:**

```

1. $matrix = array(
2.     array(1, 2, 3),

```


- **Array Functions:**

Function	Description	Example Usage
count()	Returns the number of elements in an array	count(\$fruits)
array_merge()	Merges two or more arrays	array_merge(\$arr1, \$arr2)
array_push()	Adds elements to the end of an array	array_push(\$fruits, "Grape")
array_pop()	Removes the last element from an array	array_pop(\$fruits)
in_array()	Checks if a value exists in an array	in_array("Banana", \$fruits)
array_keys()	Returns all keys of an array	array_keys(\$ages)
array_values()	Returns all values of an array	array_values(\$ages)

- **Accessing Array Elements:**

```
1. echo $fruits[0]; // Outputs: Apple
2. echo $ages['John']; // Outputs: 25
```

07. Control Structures

PHP supports the standard control structures found in most programming languages, such as conditional statements and loops.

- **If-Else Statement:**

```
1. if ($age >= 18) {
2.     echo "Adult";
3. } else {
4.     echo "Minor";
5. }
```

- **Switch Case:**

```
1. $day = "Monday";
2. switch ($day) {
3.     case "Monday":
4.         echo "It's Monday!";
5.         break;
```

- **Ternary Operator:**

```
1. $is_valid = ($age >= 18) ? true : false;
```

08. Loops

Loops in PHP allow repetitive execution of code blocks.

- **Ternary Operator:**

```
1. for ($i = 0; $i < 5; $i++) {  
2.     echo $i;  
3. }
```

- **While Loop:**

```
1. $i = 0;  
2. while ($i < 5) {  
3.     echo $i;
```

- **Do-While Loop:**

```
1. $i = 0;  
2. do {  
3.     echo $i;  
4.     $i++;  
5. } while ($i < 5);
```

- **Foreach Loop:**

```
1. $fruits = array("Apple", "Banana", "Orange");  
2. foreach ($fruits as $fruit) {  
3.     echo $fruit;  
4. }
```

09. Functions in PHP

Functions allow code reuse and modularity.

- **Defining and Calling Functions:**

```
1. function greet($name) {  
2.     echo "Hello, $name!";  
3. }  
4. greet("John"); // Outputs: Hello, John!
```

- **Return Statement:**

```
1. function add($a, $b) {  
2.     return $a + $b;  
3. }  
4. $sum = add(5, 3); // $sum is 8
```

- **Default Parameters:**

```
1. function greet($name = "Guest") {  
2.     echo "Hello, $name!";  
3. }  
4. greet(); // Outputs: Hello, Guest!
```

- **Variable-Length Argument Lists:**

```
1. function sum(...$numbers) {  
2.     return array_sum($numbers);  
3. }  
4. echo sum(1, 2, 3, 4); // Outputs: 10
```

10. Object-Oriented Programming (OOP)

PHP supports OOP concepts like classes, objects, inheritance, and polymorphism.

- **Classes and Objects:**

- A class is a blueprint for creating objects.

```
1. class Dog {  
2.     public $name;
```

- **Constructors:**

- A constructor is a special method that is automatically called when an object is created.

```
1. class Car {
2.     public $make;
3.     public function __construct($make) {
4.         $this->make = $make;
5.     }
6. }
7. $myCar = new Car("Toyota");
8. echo $myCar->make; // Outputs: Toyota
```

- **Inheritance:**

```
1. class Animal {
2.     public function makeSound() {
3.         echo "Some generic sound";
4.     }
5. }
6. class Cat extends Animal {
7.     public function makeSound() {
8.         echo "Meow!";
9.     }
10. }
11. $cat = new Cat();
12. $cat->makeSound(); // Outputs: Meow!
```

- **Access Modifiers:**

```
1. class Person {
2.     private $name;
3.     public function setName($name) {
4.         $this->name = $name;
5.     }
}
```

- **Polymorphism:**

```
1. class Shape {
2.     public function draw() {
3.         echo "Drawing a shape";
4.     }
5. }
6. class Circle extends Shape {
7.     public function draw() {
8.         echo "Drawing a circle";
9.     }
10. }
11. $shape = new Circle();
12. $shape->draw(); // Outputs: Drawing a circle
```

- **Static Methods and Properties:**

```
1. class MathHelper {
2.     public static $pi = 3.14159;
3.     public static function add($a, $b) {
4.         return $a + $b;
5.     }
6. }
7. echo MathHelper::$pi; // Outputs: 3.14159
8. echo MathHelper::add(2, 3); // Outputs: 5
```

11. Superglobals

• Constructors:

Super-global	Description
<code>\$_GET</code>	Contains data from the query string of a URL
<code>\$_POST</code>	Contains data from an HTML form submitted via POST
<code>\$_REQUEST</code>	Contains data from both GET and POST methods
<code>\$_SESSION</code>	Contains session variables
<code>\$_COOKIE</code>	Contains cookie values
<code>\$_SERVER</code>	Contains server and execution environment information
<code>\$_FILES</code>	Contains file upload information

• Using `$_GET`:

```

1. // URL: example.com/page.php?name=John
2. $name = $_GET['name'];
3. echo $name; // Outputs: John

```

• Using `$_POST`:

```

1. // HTML Form:
2. <form method="post" action="submit.php">
3.     <input type="text" name="username">
4.     <input type="submit">
5. </form>
6. <?php
7. // In submit.php
8. $username = $_POST['username'];
9. echo $username;
10. ?>

```

• Using `$_SESSION`:

- Sessions store user data across multiple page requests.

```
1. session_start();
2. $_SESSION['user'] = "John Doe";
3. echo $_SESSION['user']; // Outputs: John Doe
```

- **Using \$_COOKIE:**

- Cookies store small amounts of data on the user's machine.

```
1. setcookie("username", "John", time() + 3600); // Cookie expires in 1 hour
2. echo $_COOKIE['username']; // Outputs: John
```

12 . File Handling

PHP provides functions for reading, writing, and manipulating files.

- **Reading a File:**

```
1. $filename = "file.txt";
2. $content = file_get_contents($filename);
3. echo $content;
```

- **Writing to a File:**

```
1. $file = fopen("file.txt", "w");
2. fwrite($file, "Hello, File!");
3. fclose($file);
```

- **Checking if a File Exists:**

```
1. if (file_exists("file.txt")) {
2.     echo "File exists!";
3. }
```

- **File Uploading:**

- PHP makes file uploading simple using the \$_FILES superglobal.

```
1. <form method="post" enctype="multipart/form-data">
2.     <input type="file" name="myfile">
3.     <input type="submit">
4. </form>
5. <?php
6. if ($_FILES["myfile"]["error"] == 0) {
7.     $targetDir = "uploads/";
8.     $targetFile = $targetDir . basename($_FILES["myfile"]["name"]);
9.     move_uploaded_file($_FILES["myfile"]["tmp_name"], $targetFile);
10.    echo "File uploaded successfully!";
11. }
12. ?>
```

13 . Error Handling

Error handling in PHP is done using the try-catch block.

- **Try-Catch Block:**

```
1. try {
2.     if ($age < 18) {
3.         throw new Exception("Age must be 18 or older.");
4.     }
5. } catch (Exception $e) {
6.     echo "Error: " . $e->getMessage();
7. }
```

- **Custom Error Handler:**

- You can define a custom error handler using `set_error_handler()`.

```
1. function customError($errno, $errstr) {
2.     echo "Error: [$errno] $errstr";
3. }
4. set_error_handler("customError");
```

14 . Working with Databases

PHP provides extensions like PDO and MySQLi to work with databases.

- **Connecting to MySQL (MySQLi):**

```
1. $servername = "localhost";
2. $username = "root";
3. $password = "";
4. $dbname = "test_db";
5. $conn = new mysqli($servername, $username, $password, $dbname);
6. if ($conn->connect_error) {
7.     die("Connection failed: " . $conn->connect_error);
8. }
```

- **Querying the Database:**

```
1. $sql = "SELECT id, name FROM users";
2. $result = $conn->query($sql);
3. if ($result->num_rows > 0) {
4.     while ($row = $result->fetch_assoc()) {
5.         echo $row["id"] . " - " . $row["name"];
6.     }
7. } else {
8.     echo "0 results";
9. }
```

- **Prepared Statements (PDO):**

```
1. $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
2. $stmt = $conn->prepare("SELECT name FROM users WHERE id = ?");
3. $stmt->execute([1]);
4. $user = $stmt->fetch();
5. echo $user['name'];
```


15 .PHP Sessions and Cookies

Error handling in PHP is done using the try-catch block.

- **Session Management:**

```
1. session_start(); // Start the session
2. $_SESSION['username'] = "John"; // Store session data
3. echo $_SESSION['username']; // Access session data
```

- **Cookies Management:**

```
1. setcookie("username", "John", time() + (86400 * 30), "/"); // Set a cookie
2. echo $_COOKIE['username']; // Access the cookie
```

16 .Regular Expressions in PHP

Regular expressions are patterns used for string matching.

- **Functions for Regular Expressions:**

- **preg_match():** Searches a string for a pattern.
- **preg_replace():** Replaces occurrences of a pattern in a string.

```
1. $pattern = "/^hello/i";
2. $string = "Hello, World!";
3. if (preg_match($pattern, $string)) {
4.     echo "Pattern found!";
5. }
```

17 .PHP Date and Time

PHP provides a robust set of functions for handling dates and times.

- **Getting Current Date/Time:**

```
1. echo date("Y-m-d H:i:s"); // Outputs current date and time
```

- **Formatting Dates:**

```
1. echo date("l, F j, Y"); // Outputs: Monday, October 10, 2024
```

- **Timestamp Functions:**

```
1. $timestamp = strtotime("next Monday");  
2. echo date("Y-m-d", $timestamp); // Outputs the date of next Monday
```

18 .PHP Best Practices

- **Follow Naming Conventions:**
 - Use meaningful variable and function names.
 - Use camelCase for variables and functions, PascalCase for class names, and ALL_CAPS for constants.
- **Secure Input Validation:**
 - Use meaningful variable and function names.
 - Use camelCase for variables and functions, PascalCase for class names, and ALL_CAPS for constants.
- **Error Reporting:**
 - Use `error_reporting(E_ALL)` during development to display all types of errors.
 - In production, log errors instead of displaying them using `ini_set("log_errors", 1)` and `error_log()`.
- **Use Prepared Statements:**
 - Always use prepared statements to avoid SQL injection attacks.
- **Avoid Global Variables:**
 - Avoid using global variables as they can lead to unmanageable code. Use functions, classes, or namespaces instead.
- **Use Composer:**
 - Use Composer for managing dependencies in your PHP projects.
- **Optimize Performance:**
 - Cache expensive operations such as database queries.
 - Use object-oriented programming for better code reusability and maintenance.

19 .Conclusion

PHP is a powerful and flexible server-side language that is widely used in web development due to its ease of integration with HTML, support for databases, and the ability to handle dynamic content efficiently. This comprehensive has covered the fundamental and advanced aspects of PHP, including syntax, data types, variables, control structures, functions, object-oriented programming, file handling, error handling, sessions, and more.


As PHP continues to evolve, adopting new features and best practices will ensure your PHP applications remain secure, maintainable, and high-performing. Whether you're building a small website or a large-scale web application, PHP provides the tools you need to succeed. Happy coding!

[Click Here To Find Out More](#)

Thank you

For Your Learning Today

 elysiumacademy.org

 info@elysiumacademy.org

Scan Here for More
Spark Notes

