Internet Applications

PHP Overview

Suhel Hammoud

Introduction to PHP

Welcome to the **PHP Language Lecture Series**

For students who already know programming (Python, Java, C#, etc.)

Goals:

- Understand PHP fundamentals
- Learn syntax and runtime model
- Explore server-side web programming
- Discover modern PHP features and frameworks

What is PHP?

- **PHP** stands for Hypertext Preprocessor
- Server-side scripting language mainly used for **web development**
- Created in 1994 by Rasmus Lerdorf
- Open-source and cross-platform
- Powers over 70% of websites (WordPress, Wikipedia, etc.)

Execution Model:

- PHP code executes on the **server**
- Output (usually HTML) is sent to the **browser**

PHP in Action

PHP code can be embedded in HTML files:

```
<!DOCTYPE html>
<html>
<body>
  <h1>Example</h1>
  <?php echo "Hello, World!"; ?>
</body>
</html>
```

When a browser requests this page:

- The **PHP interpreter** executes the PHP code
- Only the **output HTML** is sent back

Setting Up PHP

Common ways to run PHP:

- **Built-in server:**

```
  php -S localhost:8000
```

- **XAMPP / MAMP / WAMP** (local development stacks)
- **Composer** for package management
- **PHP CLI** for command-line scripts

Check version:

```
php -v
```

PHP File Basics

- File extension: .php
- PHP code is enclosed with <?php ... ?>
- The closing tag is optional for pure PHP files

```php
<?php
echo "PHP works!";
```

Best practice: omit ?> in pure PHP files to avoid accidental output.

Variables and Naming

- Variables start with $
- Case-sensitive
- Must start with a letter or underscore

```php
<?php
$name = "Alice";
$Name = "Bob"; // different variable
$_count = 10;
```

Variable variables:

```php
<?php
$var = "foo";
$$var = "bar";
echo $foo; // bar
```

Constants

Define constants with `define()` or `const`.

```php
<?php
define("PI", 3.14);
const VERSION = "1.0.0";

echo PI;
```

Constants cannot start with `$` and are global by default.

Type System Overview

PHP supports **dynamic typing** but allows optional **type hints**:

```php
<?php
function add(int $a, int $b): int {
  return $a + $b;
}
echo add(2, 3);
?>
```

- **Supported scalar types:** int, float, string, bool
- **Compound:** array, object, callable
- **Special:** null, resource, mixed

Type Conversion

PHP automatically converts types when needed.

```php
<?php
$x = "10" + 2;    // 12
$y = "5 cats" * 2; // 10
```

Explicit casting:

```php
<?php
(int)$x, (float)$y, (string)$z
```

Tip: Use `var_dump()` to inspect values and types.

Operators

- Arithmetic: `+ - * / % **`
- Comparison: `==`, `===`, `!=`, `!==`, `<`, `>`
- Logical: `&&`, `||`, `!`
- String concatenation: `.`
- Ternary: `$x = $a ? $b : $c;`
- Null coalescing: `$name = $_GET['name'] ?? 'Guest';`

Control Flow Recap

PHP supports familiar control structures:

```php
<?php
if ($score >= 90) {
  echo "A";
} elseif ($score >= 80) {
  echo "B";
} else {
  echo "C";
}

switch ($day) {
  case "Mon":
    echo "Start of week";
    break;
  default:
    echo "Other day";
}
```

Loops

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

while ($condition) {
  // ...
}

do {
  // executes at least once
} while ($condition);

foreach ($array as $key => $value) {
  echo "$key => $value";
}
```

Functions: Details

- Parameters can have default values
- Use return to return values
- Functions can be nested or anonymous

```php
<?php
function power($x, $y = 2) {
  return $x ** $y;
}
echo power(3); // 9
```

Anonymous functions:

```php
<?php
$square = function($n) { return $n * $n; };
echo $square(4);
```

Arrow Functions

Introduced in PHP 7.4

```php
<?php
$double = fn($x) => $x * 2;
echo $double(10); // 20
```

Shorter syntax for single-expression closures. Lexically inherits variables like lambdas in JS/Python.

Arrays Deep Dive

Arrays in PHP are ordered maps:

- Indexed arrays
- Associative arrays

```php
<?php
$nums = [10, 20, 30];
$person = ["name" => "Alice", "age" => 30];
```

Common array functions:

- `count()`, `array_push()`, `in_array()`
- `array_keys()`, `array_values()`
- `sort()`, `asort()`, `ksort()`

Loops and Arrays Together

```php
<?php
$colors = ["red", "green", "blue"];

foreach ($colors as $index => $color) {
  echo "$index: $color\n";
}
```

Arrays can be manipulated like dictionaries or lists.

Strings Deep Dive

Concatenation: `.` Interpolation: only inside double quotes

```php
<?php
$name = "Bob";
echo "Hi $name";     // Interpolates
echo 'Hi $name';     // Literal
```

String functions:

- `strlen()`, `substr()`, `strtoupper()`, `strpos()`
- `explode()`, `implode()`, `trim()`

String Formatting

```php
<?php
printf("Name: %s, Age: %d", "Alice", 25);
```

Or using modern syntax:

```php
<?php
$name = "Alice";
echo "Hello, {$name}";
```

Multiline:

```php
<?php
$text = <<<HTML
<p>Hi $name</p>
HTML;
```

Object-Oriented PHP

PHP supports OOP features:

- Classes, inheritance, interfaces, traits
- Access modifiers (public, private, protected)

```php
<?php
class Car {
  private $brand;
  public function __construct($brand) {
    $this->brand = $brand;
  }
  public function drive() {
    echo "{$this->brand} is driving";
  }
}
```

Inheritance and Polymorphism

```php
<?php
class ElectricCar extends Car {
  public function charge() {
    echo "Charging...";
  }
}
```

Interfaces and traits help organize large projects.

Static Members

Static methods and properties belong to the class itself.

```php
<?php
class Math {
  public static $pi = 3.14;
  public static function square($n) {
    return $n * $n;
  }
}

echo Math::$pi;
echo Math::square(5);
```

Namespaces

Avoid name collisions in large codebases.

```php
<?php
namespace MyApp\Models;

class User {}
```

Use:

```php
<?php
use MyApp\Models\User;
$user = new User();
```

Error and Exception Handling

```php
<?php
try {
  throw new Exception("File not found");
} catch (Exception $e) {
  echo $e->getMessage();
} finally {
  echo "Done!";
}
```

Custom exceptions:

```php
<?php
class MyException extends Exception {}
```

File Handling

Reading and writing files:

```php
<?php
$content = file_get_contents("data.txt");
file_put_contents("log.txt", "Hello world");
```

Line-by-line:

```php
<?php
$handle = fopen("data.txt", "r");
while (($line = fgets($handle)) !== false) {
  echo $line;
}
fclose($handle);
```

Working with Forms

```html
<form method="post" action="welcome.php">
  <input name="name">
  <input type="submit">
</form>
```

```php
<?php
echo "Welcome, " . htmlspecialchars($_POST['name']);
?>
```

Always sanitize user input to avoid XSS or injection.

Sessions

Preserve user data across pages.

```php
<?php
session_start();
$_SESSION['user'] = "Alice";
echo $_SESSION['user'];
```

To destroy:

```php
<?php
session_destroy();
```

Cookies

Store small client-side data.

```php
<?php
setcookie("theme", "dark", time()+3600);
echo $_COOKIE["theme"];
```

Cookies are automatically sent with every request.

Includes and Requires

```php
<?php
include "header.php";
require "config.php";
```

Difference:

- include: Warning on missing file
- require: Fatal error on missing file

Working with JSON

```php
<?php
$data = ["name" => "Alice", "age" => 25];
$json = json_encode($data);
echo $json;

$obj = json_decode($json, true);
```

Use `true` in `json_decode()` to get an associative array.

Date and Time

```php
<?php
echo date("Y-m-d H:i:s");
$timestamp = strtotime("next Monday");
```

Use DateTime class for more power:

```php
<?php
$d = new DateTime();
echo $d->format("Y-m-d");
```

Connecting to Databases (PDO)

```php
<?php
$pdo = new PDO("mysql:host=localhost;dbname=test", "root", "");
$stmt = $pdo->query("SELECT * FROM users");
foreach ($stmt as $row) {
  echo $row['name'];
}
```

Always use **prepared statements** for safety.

Security Essentials

- Always escape output (`htmlspecialchars`)
- Use prepared statements for SQL
- Validate input types
- Never trust `$_GET` or `$_POST` directly
- Hide error details from production users

CLI PHP Scripts

PHP can run from the command line:

```php
<?php
echo "Hello from CLI\n";
```

Run with:

```
php script.php
```

Access arguments:

```php
<?php
print_r($argv);
```

Composer and Dependencies

Composer = PHP’s package manager

```
composer init
composer require guzzlehttp/guzzle
```

Autoloading:

```php
<?php
require 'vendor/autoload.php';
```

PHP 8+ Modern Features

- Union types (int|float)
- Attributes (annotations)
- Named arguments
- Match expression (like switch)
- JIT compilation

```php
<?php
$result = match($status) {
  200 => "OK",
  404 => "Not Found",
  default => "Error",
};
```

Debugging

Use:

- `var_dump()`
- `print_r()`
- `error_log()`
- Xdebug extension

Enable error reporting:

```php
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);
```

Common Pitfalls

- Forgetting `$` in variable names
- Unintended type juggling
- Unescaped user input
- Using `==` instead of `===`
- Ignoring `error_reporting`

Best Practices

- ✅ Use strict types
- ✅ Separate logic from templates
- ✅ Validate all inputs
- ✅ Use Composer
- ✅ Follow PSR standards
- ✅ Use namespaces and autoloading

```php
<?php
declare(strict_types=1);
```

PHP Ecosystem

- **Frameworks:** Laravel, Symfony, CodeIgniter
- **CMS:** WordPress, Drupal, Joomla
- **Testing:** PHPUnit
- **Templating:** Blade, Twig
- **Tools:** PHPStan, Psalm, Rector

Summary

- PHP is flexible and easy for web apps
- Syntax resembles C-style languages
- Strong ecosystem and community
- Evolving rapidly (PHP 8+)
- Ideal for full-stack web development