

Part 2 — Advanced PHP Interview Questions (Q31–Q60)

Q31. What is Autoloading in PHP?

Answer: Autoloading automatically loads classes when they are used — no need for manual `require` statements.

Example (PSR-4 Standard):

```
spl_autoload_register(function ($class) {  
    include "classes/" . $class . ".php";  
});
```

Modern PHP projects use **Composer autoloading** defined in `composer.json` under `"autoload": { "psr-4": { ...} }`.

Q32. What is Composer in PHP?

Answer: Composer is a **dependency manager** for PHP. It installs and manages external libraries with version control.

Example:

```
composer require guzzlehttp/guzzle
```

Q33. What is the difference between `self::` and `static::` in PHP?

Answer:

- `self::` → Bound to the class where the method is defined (compile-time binding).
- `static::` → Late static binding (runtime binding).

Example:

```
class A {
    public static function who() { echo __CLASS__; }
    public static function test() { static::who(); }
}
class B extends A {
    public static function who() { echo __CLASS__; }
}
B::test(); // Outputs: B
```

Q34. What is Late Static Binding?

Answer: It refers to the ability to reference the *called class* (not the defining class) in static inheritance contexts using `static::`.

Q35. What are Generators in PHP?

Answer: Generators provide a simple way to iterate large datasets without loading everything into memory.

Example:

```
function numbers() {
    for ($i = 1; $i <= 5; $i++) {
        yield $i;
    }
}
foreach (numbers() as $num) echo $num;
```

Q36. What is the difference between `include_path` and `autoload`?

- `include_path` → Static path for file inclusion.
- `autoload` → Dynamically loads classes on demand.

Q37. What are Namespaces in PHP?

Answer: Namespaces prevent naming conflicts between classes or functions in large applications.

Example:

```
namespace App\Controllers;
class UserController {}
```

Q38. What are Anonymous Functions (Closures)?

Answer: Functions without a name, often used as callbacks.

```
$greet = function($name) {
    return "Hello, $name!";
};
echo $greet("Neeraj");
```

Q39. What is a Callback Function?

Answer: A function passed as an argument to another function.

```
function process($callback) {
    $callback();
}
process(fn() => print("Done"));
```

Q40. What is a Closure Binding in PHP?

Answer: Closures can be **bound** to an object using `$closure->bindTo($object, $class)` to access its private/protected members.

Q41. What are Attributes (Annotations) in PHP 8+?

Answer: Structured metadata used to decorate classes, properties, and methods.

Example:

```
#[Route('/dashboard')]
class DashboardController {}
```

Q42. What is Reflection in PHP?

Answer: Reflection API allows runtime introspection of classes, methods, and properties — useful for frameworks and dependency injection.

Example:

```
$reflector = new ReflectionClass('User');  
print_r($reflector->getMethods());
```

Q43. What are Interfaces used for?

Answer: Interfaces define method contracts that implementing classes must follow — supports **polymorphism** and loose coupling.

Q44. What are Abstract Classes used for?

Answer: Abstract classes act as blueprints. They can have abstract (must override) and concrete (predefined) methods.

Q45. What is the difference between Interface and Abstract Class?

Feature	Interface	Abstract Class
Methods	Only declarations	Can include logic
Multiple inheritance	Yes	No
Variables	Constants only	Can have properties

Q46. What are Traits and why use them?

Answer: Traits allow sharing reusable methods between classes without using inheritance.

Q47. What is Dependency Injection (DI)?

Answer: DI is a design principle where dependencies are *passed into* a class instead of being created inside it — improves testability.

Example:

```
class Mailer {}  
class UserController {  
    public function __construct(public Mailer $mailer) {}  
}
```

Q48. What is the Repository Pattern in PHP?

Answer: It separates **data access logic** from **business logic**. The controller communicates only with the repository, not directly with the model.

Q49. What are Design Patterns commonly used in PHP?

- Singleton
 - Factory
 - Strategy
 - Observer
 - Repository
 - Dependency Injection
 - Adapter
-

Q50. What is Singleton Pattern?

Answer: Ensures only one instance of a class exists.

```
class DB {  
    private static $instance;  
    private function __construct() {}  
    public static function getInstance() {  
        return self::$instance ??= new self();  
    }  
}
```

Q51. What is Factory Pattern?

Answer: Used to create objects without specifying exact class names.

```
class ShapeFactory {
    public static function create($type) {
        return match($type) {
            'circle' => new Circle(),
            'square' => new Square(),
        };
    }
}
```

Q52. What is the Observer Pattern?

Answer: One-to-many relationship between objects — when one changes, others get notified. Used in Laravel's event-listener system.

Q53. What is the Strategy Pattern?

Answer: Encapsulates different algorithms inside interchangeable classes. Example: multiple discount strategies in an e-commerce system.

Q54. What is SPL in PHP?

Answer: SPL (Standard PHP Library) provides interfaces and classes for common data structures: `SplStack`, `SplQueue`, `SplHeap`, `SplFileObject`, etc.

Q55. What is the difference between `require_once` and `include_once`?

Answer: Both include a file only once. If the file is missing:

- `include_once` → gives a *warning*
 - `require_once` → gives a *fatal error*
-

Q56. What is Garbage Collection in PHP?

Answer: PHP uses reference counting + cycle collector to free unused memory. Manual control:

```
gc_enable();
gc_collect_cycles();
```

Q57. What are Weak References in PHP?

Answer: They allow referencing an object without preventing it from being destroyed (useful in caching).

Q58. What is OPcache in PHP?

Answer: A bytecode cache engine that stores precompiled PHP scripts in memory for faster execution.

Q59. How to handle file uploads securely in PHP?

Answer:

- Use `$_FILES` with `move_uploaded_file()`.
- Validate MIME type and file size.
- Rename uploaded files to avoid path traversal.

Example:

```
move_uploaded_file($_FILES['photo']['tmp_name'], "uploads/profile.jpg");
```

Q60. What are PHP Streams?

Answer: Streams provide a unified way to access data from files, network connections, memory, etc. Examples:

```
php://input, php://output, php://memory
```

❑ End of Part 2 (Advanced PHP)