Functions

1PHPD

1PHPD – Functions

Course Objectives

By the end of the course, students should:

- Learn about functions
- Write reusable code
- Know how to import functions

Time:

- course: 1h
- exercises: 2h

Summary

1. Functions
2. Require and Include
3. Reusable code

1. Functions

1. Functions

In PHP, a function is a block of statements that can be used repeatedly throughout a script.

Functions allow you to encapsulate a task in a single, reusable unit of code.

They are one of the fundamental building blocks of PHP programming, enabling code modularity, reusability, and readability.

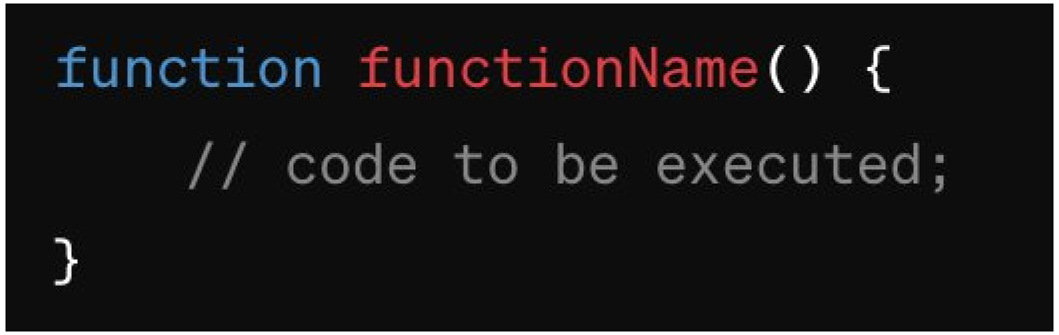
1. Functions

To define a function in PHP, you use the function keyword, followed by a unique function name, a set of parentheses, and a block of code enclosed in curly braces.

The function name is case-insensitive but follows the same naming rules as variables (must start with a letter or underscore, followed by any number of letters, numbers, or underscores).

1. Functions

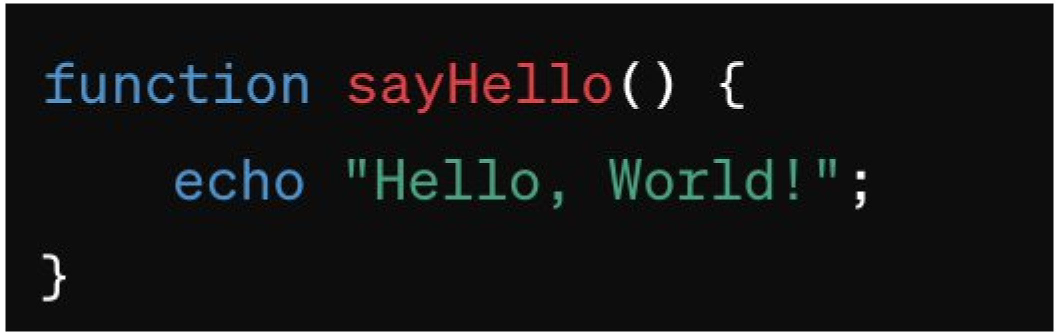
Syntax



```
function functionName() {
    // code to be executed;
}
```

1. Functions

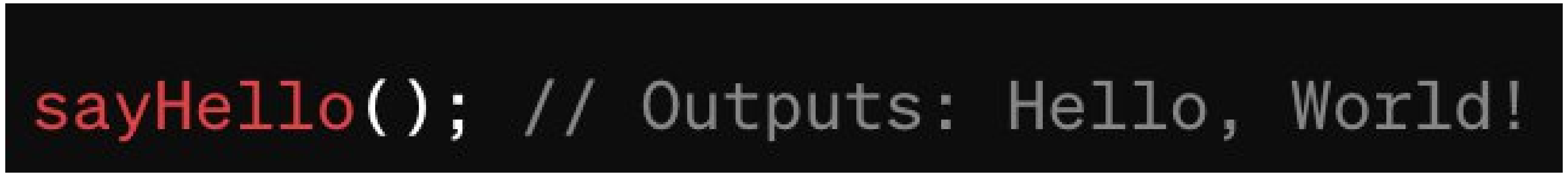
Example



```
function sayHello() {
    echo "Hello, World!";
}
```

1. Functions

To execute the code within a function, you "call" the function by using its name followed by parentheses.



```
sayHello(); // Outputs: Hello, World!
```

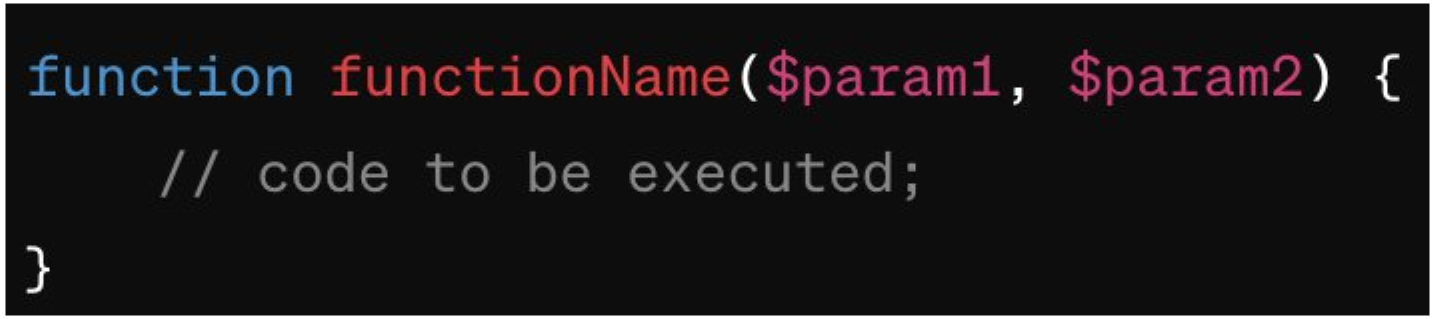
1. Functions

Functions can accept parameters (also known as arguments) that allow you to pass data into the function.

Parameters are specified within the parentheses in the function definition, and when you call the function, you provide the arguments that match those parameters.

1. Functions

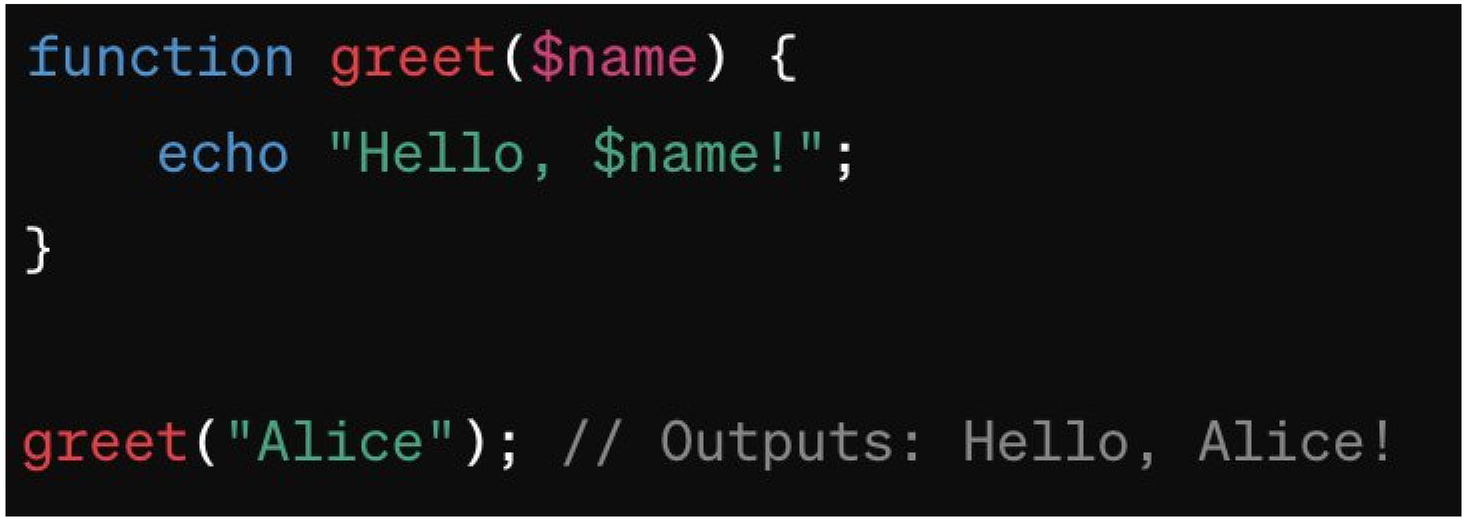
Syntax



```
function functionName($param1, $param2) {
    // code to be executed;
}
```

1. Functions

Example



```php
function greet($name) {
    echo "Hello, $name!";
}

greet("Alice"); // Outputs: Hello, Alice!
```

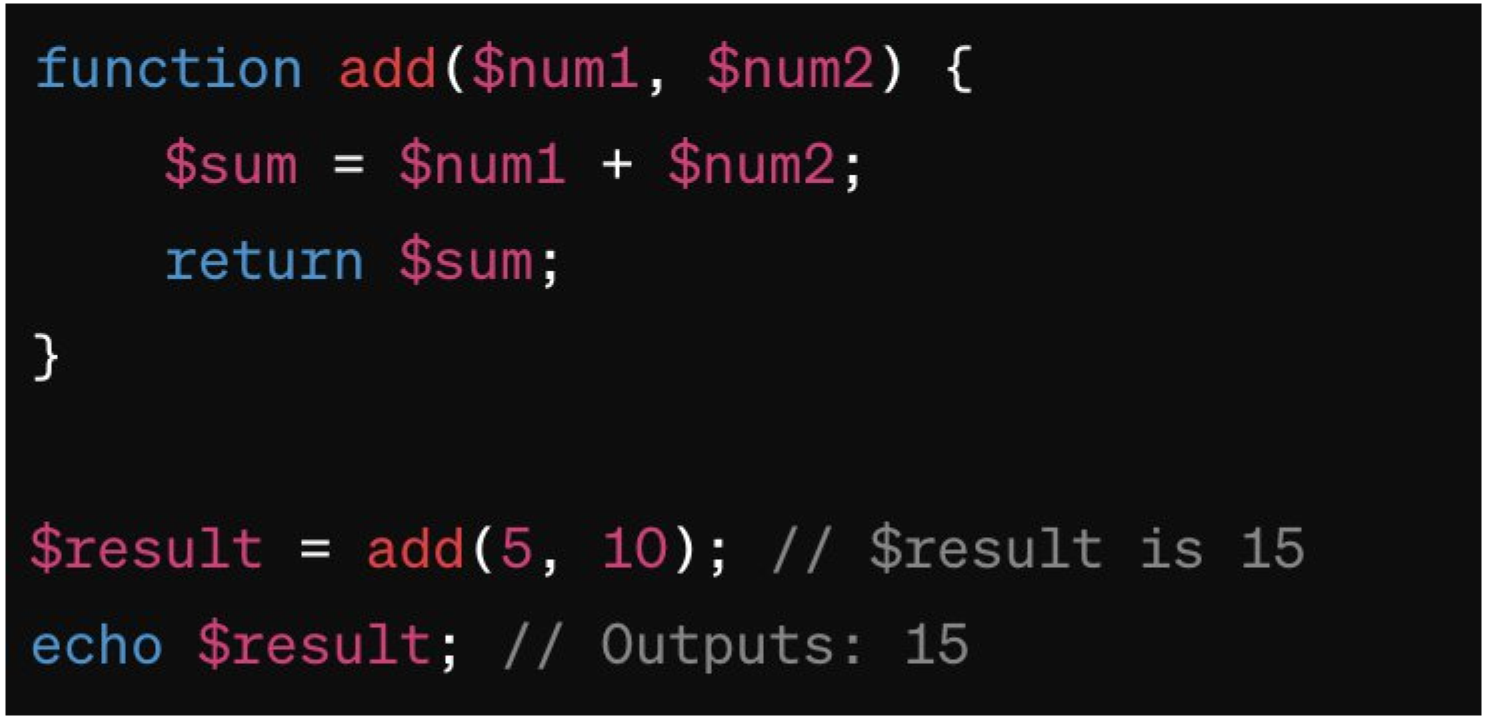
1. Functions

Functions can return values to the calling code using the return statement.

Once a return statement is executed, the function ends and returns control to the calling code, along with the return value (if any).

1. Functions

Example



```
function add($num1, $num2) {
    $sum = $num1 + $num2;
    return $sum;
}

$result = add(5, 10); // $result is 15
echo $result; // Outputs: 15
```

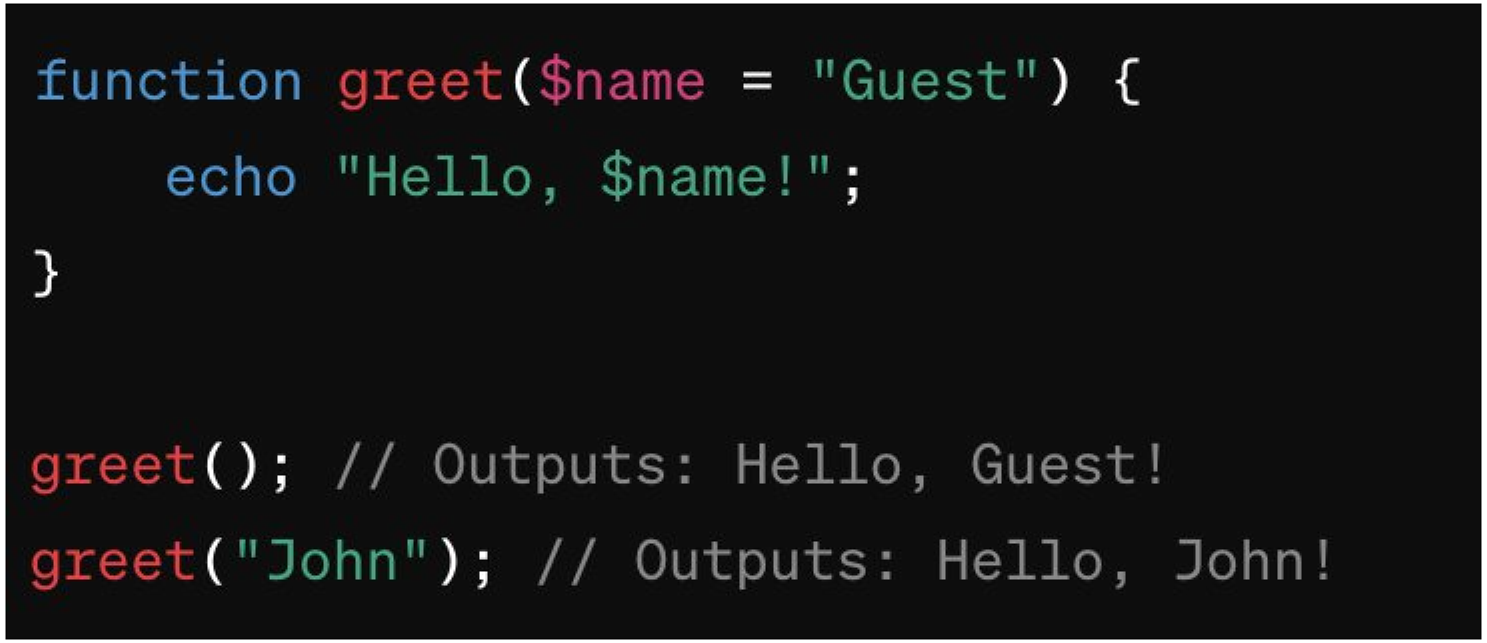
1. Functions

You can assign default values to function parameters.

If an argument for that parameter is not provided when the function is called, the default value is used.

1. Functions

Example



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

greet(); // Outputs: Hello, Guest!
greet("John"); // Outputs: Hello, John!
```

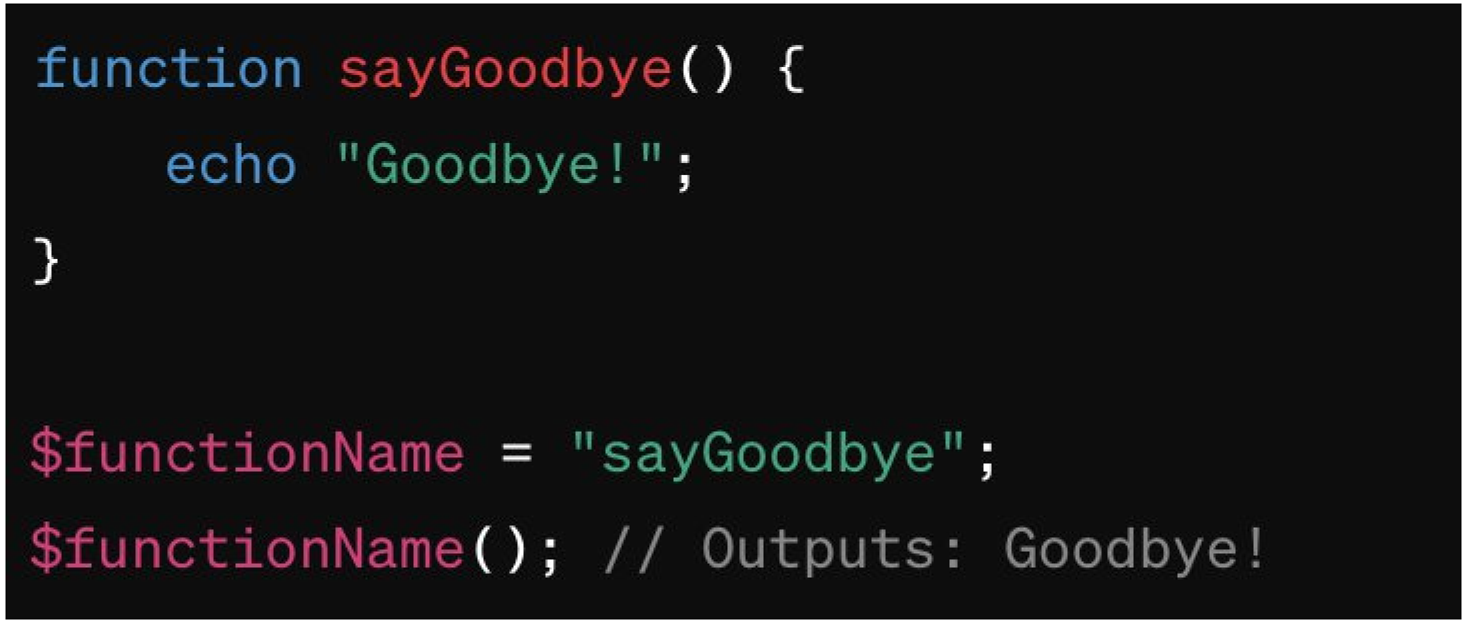
1. Functions

PHP supports variable functions, meaning you can store the name of a function within a variable and then call the function using this variable.

This can be useful in some specific cases, but it is not recommended to use that solution as a default solution to store your functions.

1. Functions

Example



```php
function sayGoodbye() {
    echo "Goodbye!";
}

$functionName = "sayGoodbye";
$functionName(); // Outputs: Goodbye!
```

1. Functions

PHP already provides a lot of built-in functions, some of them that you know already

- the functions used to create loops (for, foreach …)
- echo to display some values
- the functions to display the content of an array

1. Functions

Functions are a core concept in PHP and many other programming languages.

They provide a powerful way to encapsulate logic, making code easier to understand, maintain, and debug.

Through the use of functions, PHP developers can write more modular and efficient code.

Exercises

2. Require and Include

2. Require and Include

In PHP, include and require statements are used to insert the content of one PHP file into another PHP file before the server executes it.

This allows for code modularity, enabling the reuse of PHP code across multiple scripts, which is a fundamental aspect of developing maintainable and scalable applications.

Despite their similarities, there's a critical difference in how they handle failures.

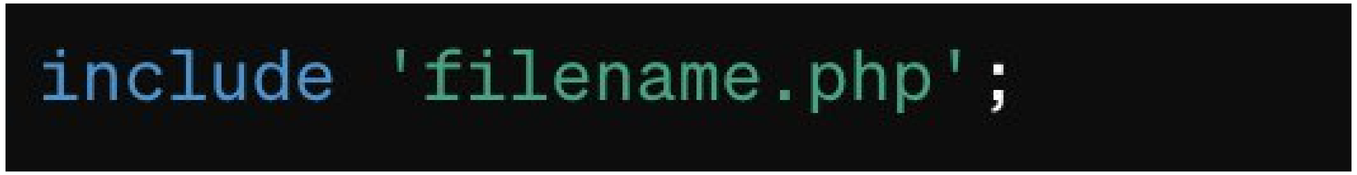
2. Require and Include

The include statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

If the file being included is not found, PHP will emit a warning (E_WARNING), but the script will continue to execute.

2. Require and Include

Syntax



```
include 'filename.php';
```

2. Require and Include

Use include when the file is not essential to the application running.

For example, including a header, footer, or menu file where, if not found, the rest of the page can still load and function properly.

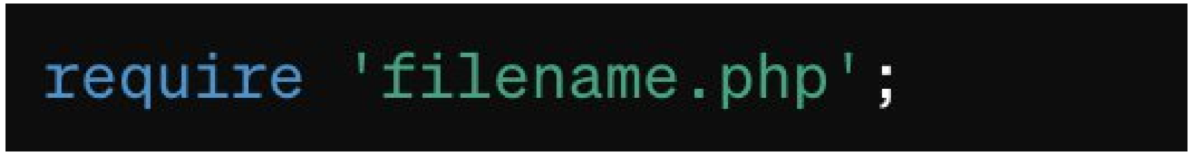
2. Require and Include

The require statement is also used to include the content of one PHP file into another.

It behaves exactly like include, except upon failure, it produces a fatal error (E_COMPILE_ERROR) and stops the script execution.

2. Require and Include

Syntax



```php
require 'filename.php';
```

2. Require and Include

Use require when the file is essential for the application to run.

For example, loading a file that contains a database connection script or crucial application logic that the rest of the script depends on.

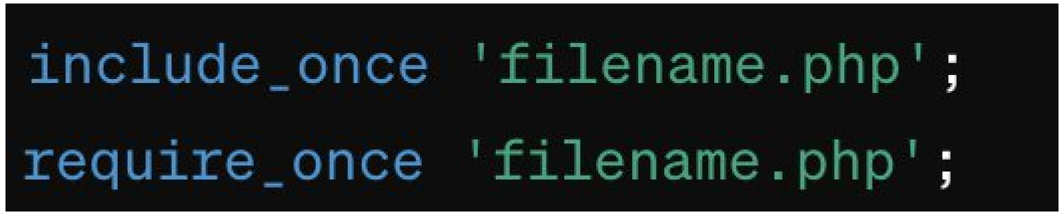
2. Require and Include

PHP also offers include_once and require_once statements.

These statements are identical to include and require respectively, except PHP will check if the file has already been included, and if so, it will not include (require) it again.

2. Require and Include

Example



```php
include_once 'filename.php';
require_once 'filename.php';
```

2. Require and Include

These are useful for including files where you need to ensure that the file is included only once and further calls to include or require it are ignored.

This is particularly important for files that define classes, functions, and other constructs to prevent redefinition errors.

2. Require and Include

Choosing Between include and require

- Use include or include_once when the file is not critical to the application's functionality, and its absence would not necessarily break the application.
- Use require or require_once when the file is essential to the application, and its absence would prevent the application from running correctly.

2. Require and Include

In summary, include and require are powerful tools for PHP developers, promoting code reuse and modularity.

Choosing between them depends on how critical the included file is to the application, ensuring error handling is managed appropriately.

2. Require and Include

For high quality code, it is recommended to only use “require_once”. This way you won’t have an issue in production by loading two time the same component (using include) or missing some file that are not critical but serve a visual purpose.

By hardening your code quality from the start, there is less risks of eros when you reach production.

Exercises

3. Reusable code

3. Reusable code

Reusable code in PHP, as in any programming language, refers to writing code in a way that it can be used multiple times throughout a project or even across multiple projects without needing to be rewritten.

Reusability is a key principle of efficient software development, enabling developers to save time, reduce errors, and improve maintainability.

3. Reusable code

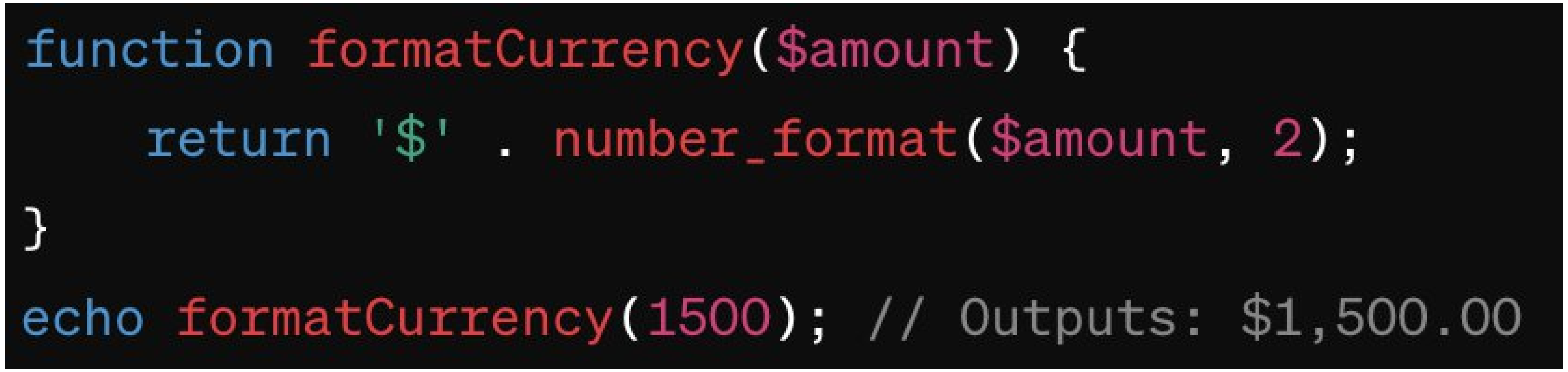
Functions

Functions encapsulate a block of code to perform a specific task and can be called multiple times with different inputs.

By defining functions for tasks that need to be performed repeatedly, you create reusable code blocks.

3. Reusable code

Example



```php
function formatCurrency($amount) {
    return '$' . number_format($amount, 2);
}
echo formatCurrency(1500); // Outputs: $1,500.00
```

3. Reusable code

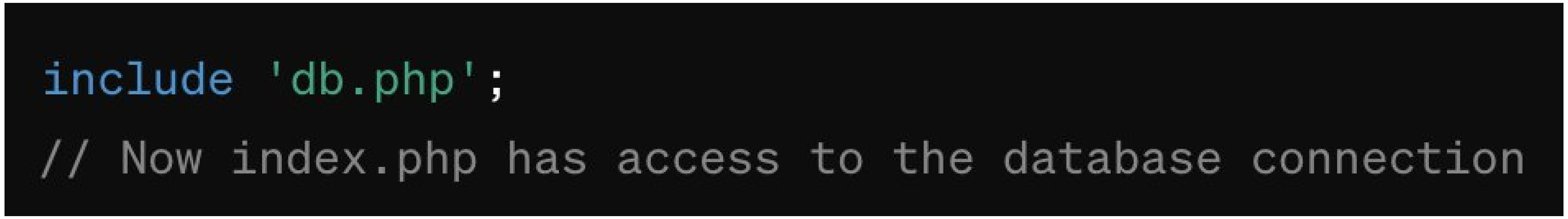
Include and Require Statements

PHP's include and require statements allow you to insert the content of one PHP file into another.

This means you can write common functionalities, like database connection scripts, header, footer, or navigation menu, in separate files and include them wherever needed.

3. Reusable code

Example



```php
include 'db.php';
// Now index.php has access to the database connection
```

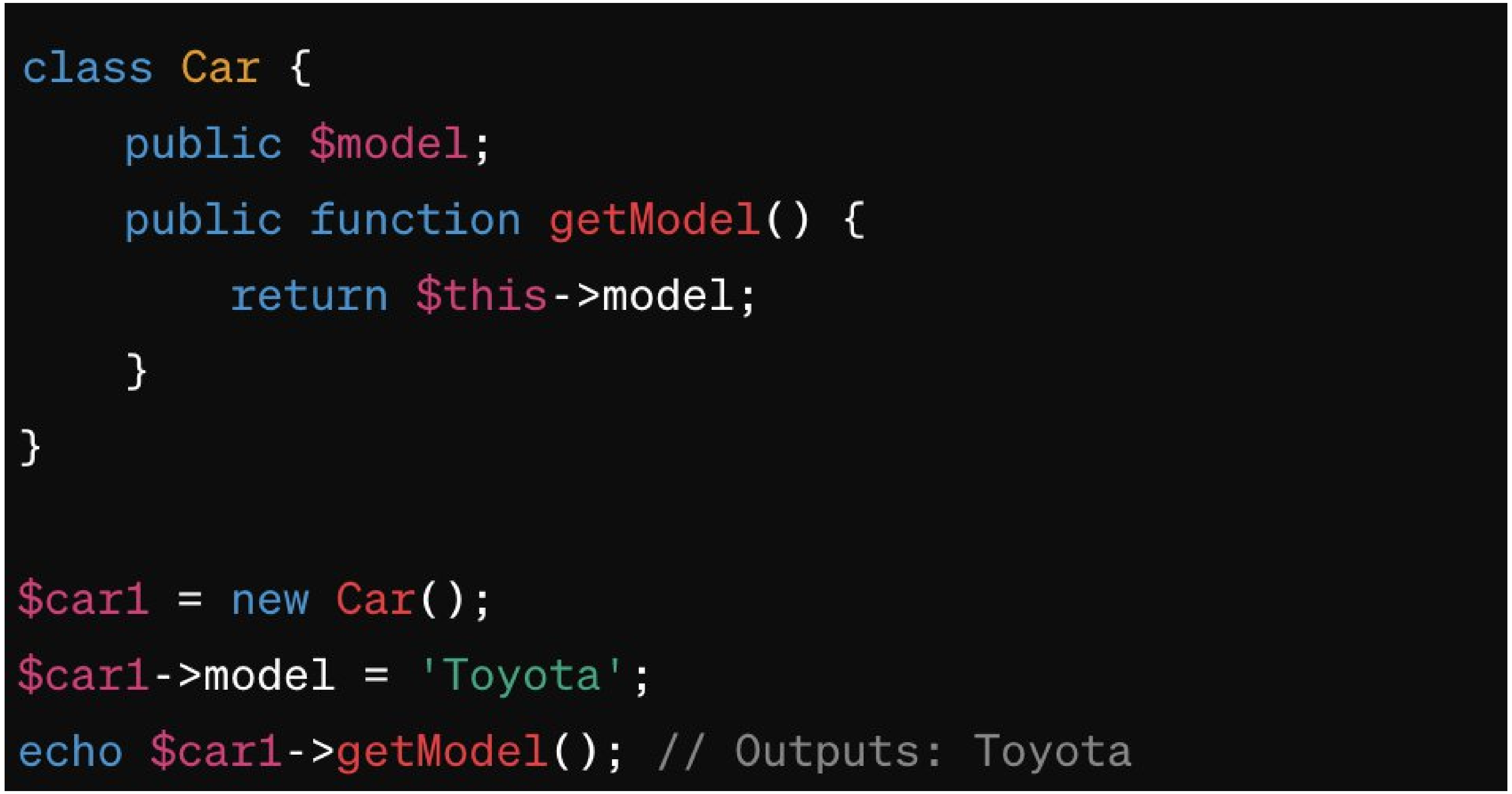
3. Reusable code

Object-Oriented Programming (OOP) in PHP allows you to define classes which are blueprints for objects.

Classes can encapsulate data and functionalities together as methods and properties, making them reusable across different parts of a project or even different projects.

3. Reusable code

Example



```php
class Car {
    public $model;
    public function getModel() {
        return $this->model;
    }
}

$car1 = new Car();
$car1->model = 'Toyota';
echo $car1->getModel(); // Outputs: Toyota
```

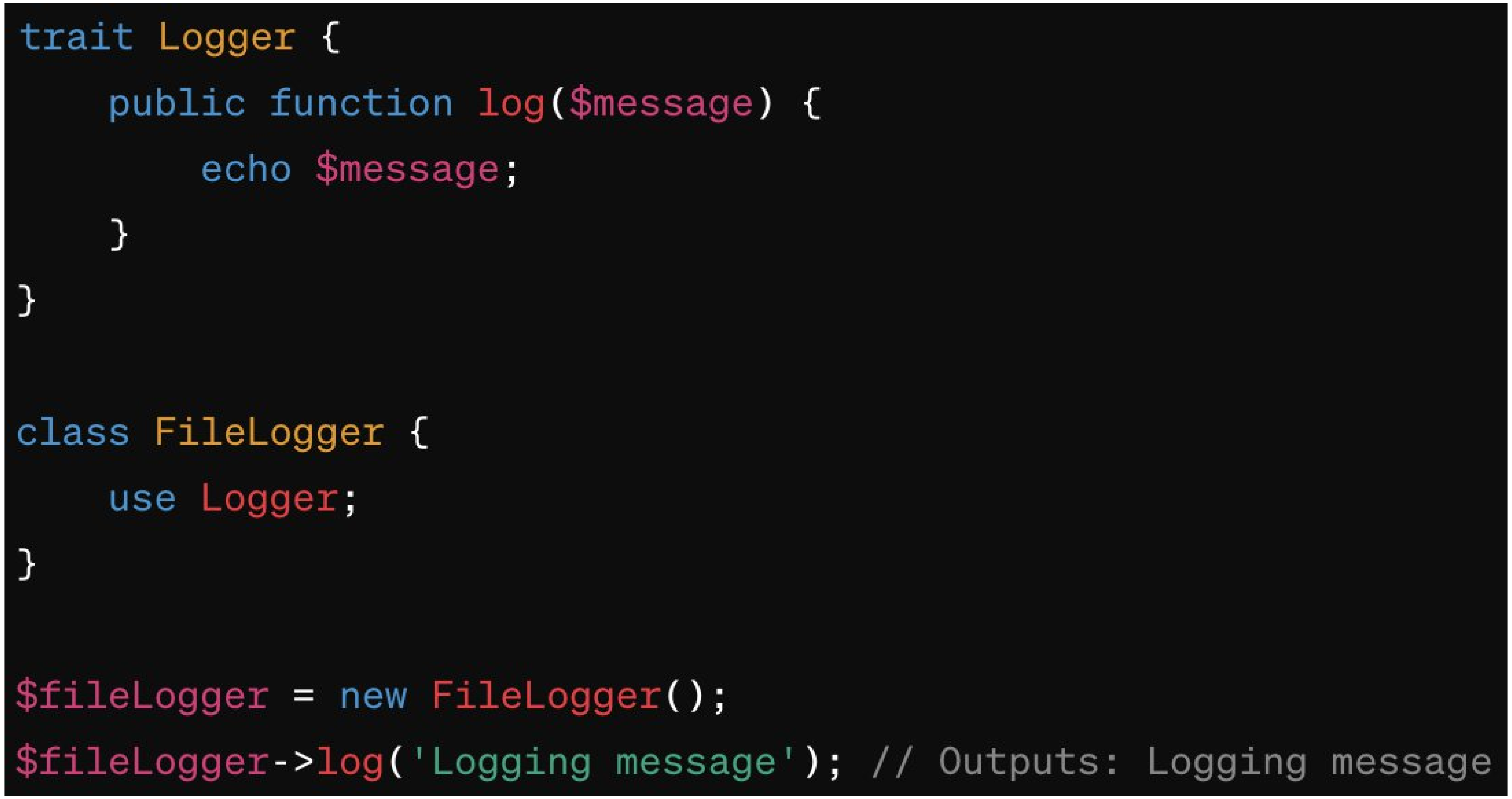
3. Reusable code

Traits are a mechanism for code reuse in single inheritance languages like PHP.

A Trait is intended to reduce some limitations of single inheritance by enabling a developer to reuse sets of methods freely in several independent classes.

3. Reusable code

Example



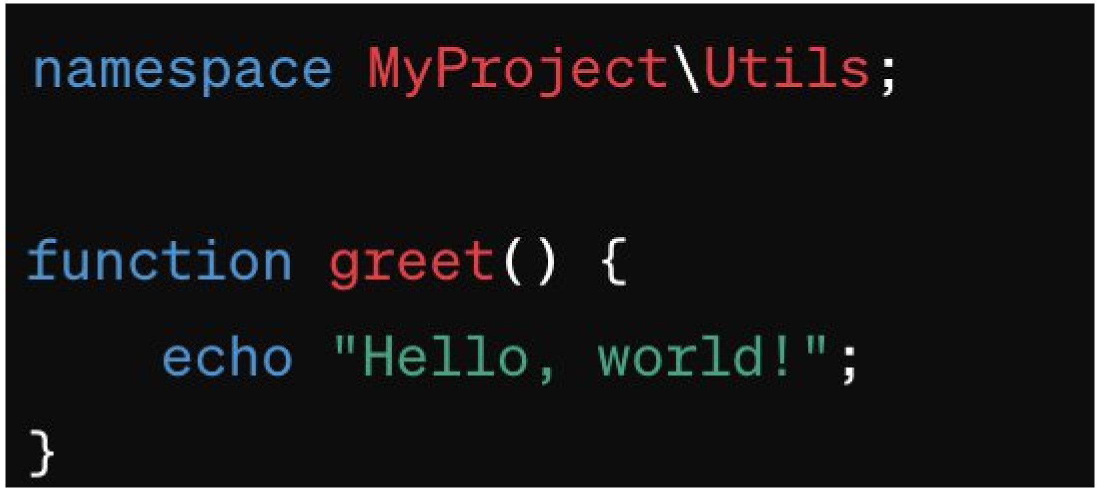
```
trait Logger {
    public function log($message) {
        echo $message;
    }
}

class FileLogger {
    use Logger;
}

$fileLogger = new FileLogger();
$fileLogger->log('Logging message'); // Outputs: Logging message
```

3. Reusable code

Namespaces are a way of encapsulating items such as functions, classes, and constants to avoid name conflicts and make it easier to create reusable code libraries.

3. Reusable code

Example

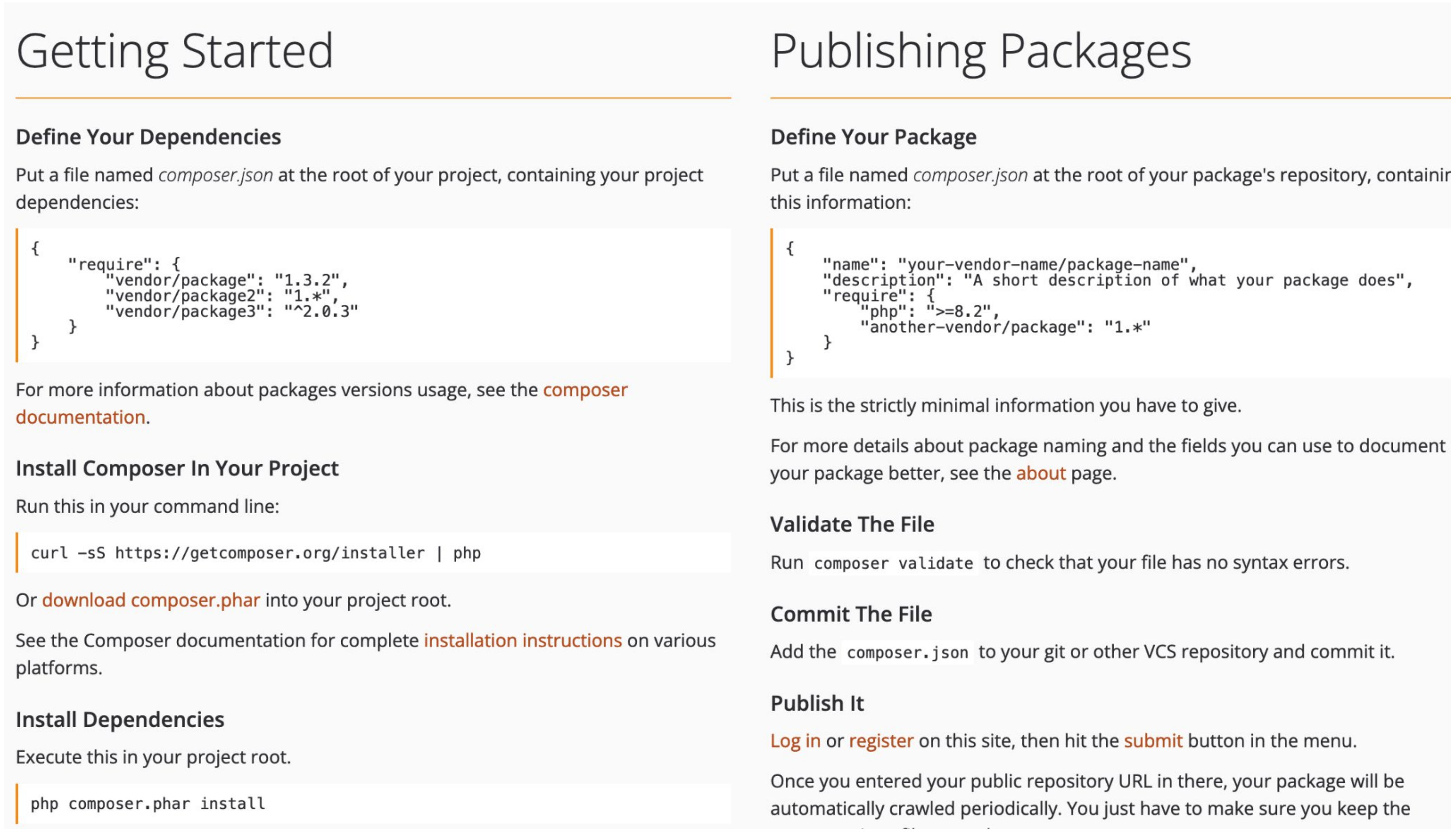

```php
namespace MyProject\Utils;

function greet() {
    echo "Hello, world!";
}
```

3. Reusable code

For larger scale reusability, PHP has Composer, a dependency manager that allows you to use and manage libraries in your projects.

Packagist is the main repository for PHP packages that can be managed with Composer, offering a vast ecosystem of reusable code.

3. Reusable code



Getting Started

Define Your Dependencies

Put a file named *composer.json* at the root of your project, containing your project dependencies:

```
{
    "require": {
        "vendor/package": "1.3.2",
        "vendor/package2": "1.*",
        "vendor/package3": "^2.0.3"
    }
}
```

For more information about packages versions usage, see the composer documentation.

Install Composer In Your Project

Run this in your command line:

```
curl -sS https://getcomposer.org/installer | php
```

Or download composer.phar into your project root.

See the Composer documentation for complete installation instructions on various platforms.

Install Dependencies

Execute this in your project root.

```
php composer.phar install
```

Publishing Packages

Define Your Package

Put a file named *composer.json* at the root of your package's repository, containir
this information:

```
{
    "name": "your-vendor-name/package-name",
    "description": "A short description of what your package does",
    "require": {
        "php": ">=8.2",
        "another-vendor/package": "1.*"
    }
}
```

This is the strictly minimal information you have to give.

For more details about package naming and the fields you can use to document your package better, see the about page.

Validate The File

Run `composer validate` to check that your file has no syntax errors.

Commit The File

Add the `composer.json` to your git or other VCS repository and commit it.

Publish It

Log in or register on this site, then hit the submit button in the menu.

Once you entered your public repository URL in there, your package will be automatically crawled periodically. You just have to make sure you keep the

3. Reusable code

By utilizing these features and strategies, PHP developers can write code that is more modular, easier to maintain, and can be reused across multiple projects, significantly speeding up the development process and improving software quality.

Exercises