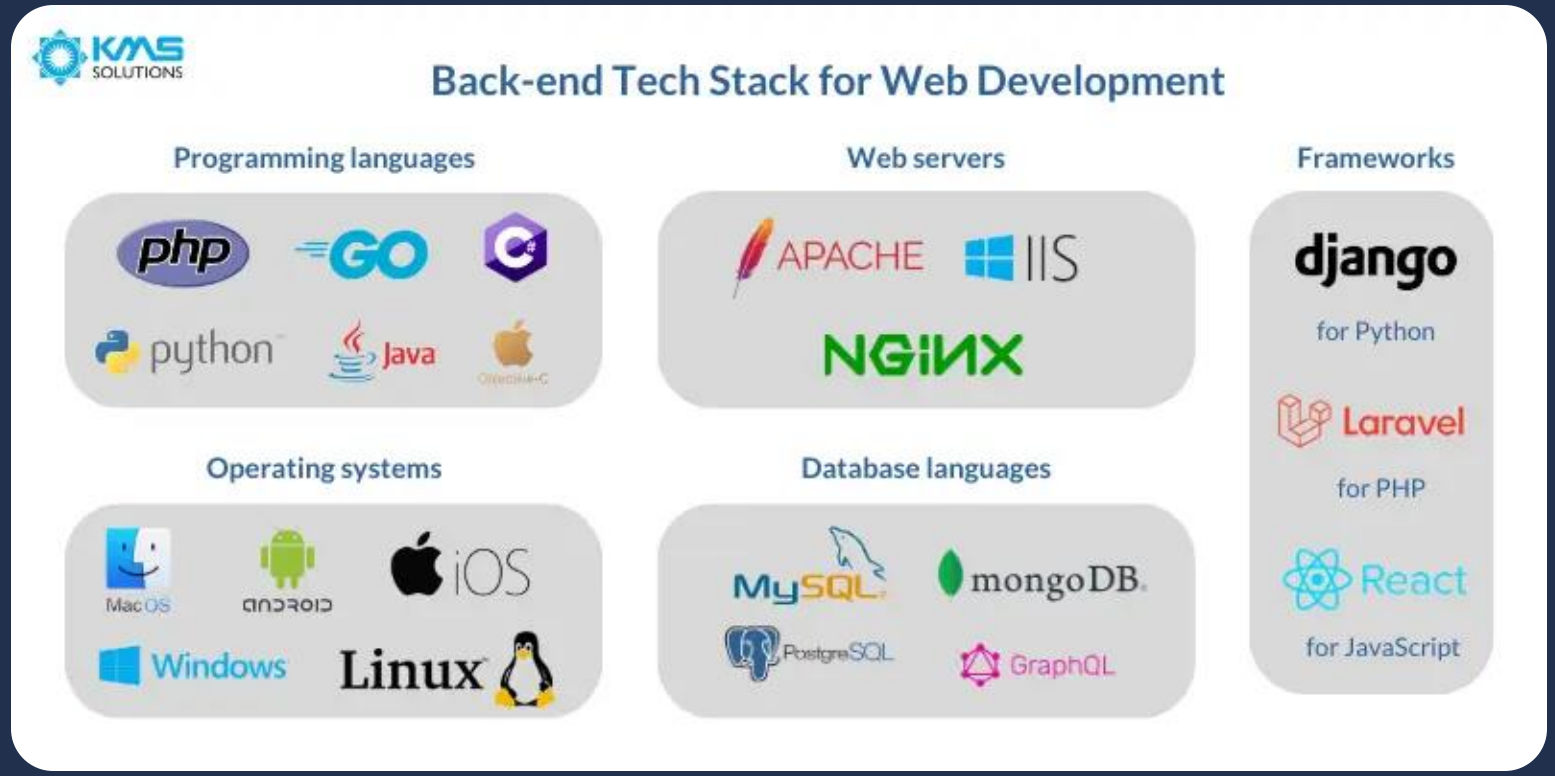
CSE485 – Công nghệ Web

dungkt@tlu.edu.vn



Bài 2. Lập trình PHP căn bản

NỘI DUNG

1. Biến, biểu thức & toán tử
2. Cấu trúc điều khiển
3. Hàm (functions)
4. Đối tượng và lớp (objects & classes)

1. Biến, biểu thức & toán tử

- **Tạo và truy cập biến**:
  - **Định nghĩa Biến**: Trong PHP, biến được định nghĩa bằng cách sử dụng dấu $ theo sau là tên của biến. Tên biến không được bắt đầu bằng số và không được chứa khoảng trắng.

```
$name = "John Doe";
$age = 30;
```

  - **Kiểu dữ liệu**: PHP là ngôn ngữ lập trình có kiểu động, nghĩa là bạn không cần phải khai báo kiểu dữ liệu của biến. PHP tự động xác định kiểu dữ liệu dựa trên giá trị được gán.

```
$number = 5;         // Kiểu số nguyên
$text = "Hello";     // Kiểu chuỗi
$floating = 5.75;    // Kiểu số thực
```

  - **Truy cập biến**:
    - Đọc giá trị: Để truy cập giá trị của biến, bạn chỉ cần sử dụng tên biến.

```
echo $name;  // In ra "John Doe"
```

    - Sửa đổi giá trị: Bạn có thể thay đổi giá trị của biến bằng cách gán một giá trị mới cho nó.

```
$age = 31;  // Cập nhật giá trị của $age
```

    - Kết hợp chuỗi: Bạn có thể kết hợp chuỗi với biến để tạo ra một chuỗi mới.

```
$greeting = "Hello, my name is " . $name; // Kết hợp chuỗi
echo $greeting;  // In ra "Hello, my name is John Doe"
```

1. Biến, biểu thức & toán tử

- **Tạo và truy cập mảng kết hợp**:
  - **Định nghĩa** mảng kết hợp (associative array) là một loại mảng mà trong đó mỗi phần tử được xác định bởi một khóa (key) cụ thể thay vì chỉ số (index). Dưới đây là cách tạo và truy cập mảng kết hợp trong PHP:

  - **Khởi tạo mảng kết hợp**: Bằng cách gán một mảng với các cặp khóa-giá trị.

```
$person = array(
    "name" => "John Doe",
    "age" => 30,
    "email" => "johndoe@example.com"
);
```

```
$person = [
    "name" => "John Doe",
    "age" => 30,
    "email" => "johndoe@example.com"
];
```

  - Truy cập phần tử của mảng kết hợp

```
echo $person["name"];  // In ra "John Doe“
$person["age"] = 31;  // Cập nhật tuổi
$person["city"] = "New York";  // Thêm thành phố
```

1. Biến, biểu thức & toán tử

- **Tạo và truy cập mảng kết hợp**:
  - **Duyệt mảng kết hợp**: Bằng cách gán một mảng với các cặp khóa-giá trị.

```
foreach ($person as $key => $value) {
    echo $key . ": " . $value . "<br>";
}
```

  - **Lưu ý**:
    - Kiểu dữ liệu của khóa: trong mảng kết hợp, khóa có thể là một chuỗi hoặc một số nguyên.
    - Trường hợp khóa trùng lặp: nếu có hai phần tử trong mảng cùng có khóa giống nhau, phần tử sau sẽ ghi đè lên phần tử trước.

1. Biến, biểu thức & toán tử

- **Tạo và truy cập mảng chỉ mục:**
  - **Định nghĩa** mảng được lập chỉ mục (indexed array) là loại mảng mà trong đó mỗi phần tử được truy cập thông qua một chỉ số (index) số nguyên. Dưới đây là cách tạo và truy cập mảng được lập chỉ mục:

  - **Khởi tạo mảng chỉ mục**: Mảng được lập chỉ mục bằng cách sử dụng hàm array() hoặc sử dụng cú pháp ngắn gọn [].

```
$colors = array("red", "green", "blue");
```

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

  - **Truy cập phần tử của mảng chỉ mục**

```
echo $colors[0];  // In ra "red"
$colors[0] = "yellow";  // Thay đổi giá trị của phần tử đầu tiên
$colors[] = "purple";  // Thêm "purple" vào cuối mảng
```

1. Biến, biểu thức & toán tử

- **Tạo và truy cập mảng chỉ mục**:
  - **Duyệt mảng chỉ mục**: Sử dụng vòng lặp foreach hoặc for để duyệt qua mỗi phần tử trong mảng.

```
foreach ($colors as $color) {
    echo $color . "<br>";
}
```

```
for ($i = 0; $i < count($colors); $i++) {
    echo $colors[$i] . "<br>";
}
```

  - **Lưu ý**:
    - Chỉ số khởi đầu: trong PHP, chỉ số của mảng được lập chỉ mục luôn bắt đầu từ 0.
    - Loại bỏ phần tử: bạn có thể loại bỏ một phần tử khỏi mảng sử dụng hàm unset().

1. Biến, biểu thức & toán tử

- **Mảng đa chiều:**
  - **Khởi tạo:**

```
$matrix = array(
    array(1, 2, 3),
    array(4, 5, 6),
    array(7, 8, 9)
);
```

```
$matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];
```

1. Biến, biểu thức & toán tử

- Sử dụng cách viết ngắn gọn echo:
  - Trong PHP 8, cũng như các phiên bản trước đó, bạn vẫn có thể sử dụng cách viết ngắn gọn (shorthand) của lệnh echo để in nội dung. Cách viết này đặc biệt hữu ích khi bạn đang làm việc với PHP trong các tệp HTML.

```
<?= $variable; ?>
thay vì
<?php echo $variable; ?>
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Demo</title>
</head>
<body>
    <p>Giá trị của biến là: <?= $variable; ?></p>
</body>
</html>
```

1. Biến, biểu thức & toán tử

- **Sử dụng toán tử số học:**
  - Các toán tử số học được sử dụng để thực hiện các phép toán cơ bản như cộng, trừ, nhân, chia, và phép lấy phần dư. Dưới đây là một số ví dụ về cách sử dụng các toán tử số học trong PHP:

```
$number1 = 12;
$number2 = 5;

$sum = $number1 + $number2;          // Tổng
$difference = $number1 - $number2;   // Hiệu
$product = $number1 * $number2;      // Tích
$quotient = $number1 / $number2;     // Thương
$remainder = $number1 % $number2;    // Phần dư

echo "Tổng: $sum\n";
echo "Hiệu: $difference\n";
echo "Tích: $product\n";
echo "Thương: $quotient\n";
echo "Phần dư: $remainder\n";
```

1. Biến, biểu thức & toán tử

- **Sử dụng toán tử so sánh:**
  - Toán tử so sánh được sử dụng để so sánh giữa hai giá trị. Dựa trên kết quả của phép so sánh, chúng trả về một giá trị Boolean (true hoặc false). Các toán tử so sánh này là nền tảng của việc kiểm tra điều kiện và quyết định logic trong các chương trình:

```
if (3 == "3") {
    echo "True"; // Điều này là true vì "3" được chuyển đổi thành số.
}
```

```
if (3 === "3") {
    echo "True";
} else {
    echo "False"; // Điều này là false vì "3" là chuỗi và 3 là số nguyên.
}
```

  - Lưu ý:
    - Phân biệt giữa == và ===: == chỉ so sánh giá trị, trong khi === còn so sánh cả kiểu dữ liệu. Tương tự, != và !== cũng có sự khác biệt tương tự.
    - Chuyển đổi kiểu dữ liệu tự động: trong php, khi sử dụng == hoặc !=, có thể xảy ra hiện tượng chuyển đổi kiểu dữ liệu tự động, điều này có thể dẫn đến kết quả không mong muốn trong một số trường hợp.

1. Biến, biểu thức & toán tử

- **Sử dụng toán tử logic**
  - Toán tử logic được sử dụng để kết hợp các biểu thức điều kiện. Chúng thường được sử dụng trong các cấu trúc điều khiển như câu lệnh if, vòng lặp while, v.v., để kiểm tra nhiều điều kiện cùng một lúc. Dưới đây là các toán tử logic cơ bản trong PHP:

```
$age = 20;
$is_student = true;

if ($age >= 18 && $is_student) {
    echo "Người trưởng thành và là sinh viên";
} elseif ($age >= 18 || $is_student) {
    echo "Người trưởng thành hoặc là sinh viên";
} else {
    echo "Không đáp ứng điều kiện";
}
```

1. Biến, biểu thức & toán tử

- **Nối chuỗi trong PHP:**
  - Việc nối chuỗi HTML với các biến là một kỹ thuật phổ biến, đặc biệt khi bạn muốn tạo nội dung động cho trang web của mình. Dưới đây là một số cách để thực hiện điều này::

```php
$name = "Alice";
$htmlString = "<p>Hello, " . $name . "!</p>";

echo $htmlString; // In ra: <p>Hello, Alice!</p>
```

```php
$name = "Alice";
echo "<p>Hello, ", $name, "!</p>"; // In ra: <p>Hello, Alice!</p>
```

```php
$name = "Alice";
?>
<p>Hello, <?= $name ?>!</p>
<?php
```

```php
$name = "Alice";
$htmlString = <<<HTML
<p>Hello, $name!</p>
HTML;

echo $htmlString; // In ra: <p>Hello, Alice!</p>
```

1. Biến, biểu thức & toán tử

```
$products = [
    ['name' => 'Sản phẩm 1', 'price' => 1000],
    ['name' => 'Sản phẩm 2', 'price' => 2000],
    ['name' => 'Sản phẩm 3', 'price' => 3000]
];
```

- **Kết hợp PHP và HTML:**
  - Ví dụ thực tế

Tách biệt logic và hiển thị: trong các ứng dụng lớn hơn, việc tách biệt logic ứng dụng (trong php) và giao diện người dùng (trong html) là quan trọng để dễ dàng bảo trì và quản lý mã nguồn.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Danh Sách Sản Phẩm</title>
</head>
<body>
    <h1>Danh Sách Sản Phẩm</h1>
    <table border="1">
        <tr>
            <th>Tên Sản Phẩm</th>
            <th>Giá</th>
        </tr>

        <?php foreach ($products as $product): ?>
            <tr>
                <td><?= htmlspecialchars($product['name']) ?></td>
                <td><?= htmlspecialchars($product['price']) ?> VND</td>
            </tr>
        <?php endforeach; ?>
    </table>
</body>
</html>
```

1. Biến, biểu thức & toán tử

```
$currentYear = date("Y"); // Lấy năm hiện tại
$years = range(1990, $currentYear); // Tạo mảng các năm từ 1990 đến năm hiện tại
```

- **Kết hợp PHP và HTML:**
  - Ví dụ thực tế

Tách biệt logic và hiển thị: trong các ứng dụng lớn hơn, việc tách biệt logic ứng dụng (trong php) và giao diện người dùng (trong html) là quan trọng để dễ dàng bảo trì và quản lý mã nguồn.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Chọn Năm</title>
</head>
<body>
    <h1>Chọn Năm</h1>
    <form action="">
        <label for="year">Năm:</label>
        <select name="year" id="year">
            <?php foreach ($years as $year): ?>
                <option value="<?= $year ?>"><?= $year ?></option>
            <?php endforeach; ?>
        </select>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

2. Các cấu trúc điều khiển

- **Cấu trúc chứa IF:**
  - if

```
$age = 20;

if ($age > 18) {
    echo "Bạn là người trưởng thành.";
}
```

  - if ... else

```
$age = 16;

if ($age > 18) {
    echo "Bạn là người trưởng thành.";
} else {
    echo "Bạn vẫn còn nhỏ.";
}
```

2. Các cấu trúc điều khiển

- **Cấu trúc chứa IF:**
  - if .. else if .. Else

```
$age = 20;

if ($age < 13) {
    echo "Bạn là trẻ em.";
} elseif ($age < 18) {
    echo "Bạn là thanh thiếu niên.";
} else {
    echo "Bạn là người trưởng thành.";
}
```

  - Toán tử 3 ngôi

```
$age = 20;
$category = ($age < 18) ? 'child' : 'adult';

echo $category; // In ra: adult
```

2. Các cấu trúc điều khiển

- **Cấu trúc với SWITCH:**

```
$day = "Mon";

switch ($day) {
    case "Mon":
        echo "Hôm nay là thứ Hai";
        break;
    case "Tue":
        echo "Hôm nay là thứ Ba";
        break;
    case "Wed":
        echo "Hôm nay là thứ Tư";
        break;
    // Thêm các trường hợp khác nếu cần
    default:
        echo "Không xác định";
}
```

```
$color = "red";

switch ($color) {
    case "red":
    case "blue":
    case "green":
        echo "Màu sắc là một màu cơ bản";
        break;
    default:
        echo "Màu sắc không phải là một màu cơ
bản";
}
```

2. Các cấu trúc điều khiển

- **Cấu trúc với MATCH:**
  - Kể từ phiên bản PHP 8.0, một tính năng mới có tên là "Match Expression" đã được giới thiệu. Đây là một cải tiến của cấu trúc switch, mang lại cách viết mã ngắn gọn và rõ ràng hơn khi bạn cần so sánh một biến với nhiều giá trị khác nhau.

```
$dayOfWeek = 1;

$dayName = match ($dayOfWeek) {
    1 => 'Monday',
    2 => 'Tuesday',
    3 => 'Wednesday',
    4 => 'Thursday',
    5 => 'Friday',
    6 => 'Saturday',
    7 => 'Sunday',
    default => 'Unknown'
};

echo $dayName; // In ra: Monday
```

```
$color = 'red';

$description = match ($color) {
    'red', 'blue', 'green' => 'This is a primary
color',
    default => 'This is not a primary color'
};

echo $description; // In ra: This is a primary
color
```

- So sánh nghiêm ngặt: khác với switch, match thực hiện so sánh nghiêm ngặt (===), nghĩa là nó so sánh cả giá trị lẫn kiểu dữ liệu.
- Trả về giá trị: match không chỉ là một cấu trúc điều khiển, mà nó còn trả về giá trị. Do đó, bạn có thể gán giá trị trả về của match cho một biến hoặc sử dụng trực tiếp.
- Cần có default: trong match, bạn cần phải xử lý trường hợp default. Nếu không có default và không có giá trị nào khớp, PHP sẽ ném ra một ngoại lệ.
- Không có break: không cần sử dụng break trong match như trong switch, vì mỗi nhánh tự động kết thúc sau khi thực thi.

2. Các cấu trúc điều khiển

- **Cấu trúc với WHILE:**
  - while là một cấu trúc điều khiển cơ bản dùng để thực hiện một khối mã nhiều lần miễn là một điều kiện nhất định được thỏa mãn. Vòng lặp while rất hữu ích khi bạn không biết trước số lần cần lặp qua một đoạn mã.

```
$i = 1;

while ($i <= 10) {
    echo $i . " ";
    $i++;
}
```

```
$colors = ["red", "green", "blue"];
$i = 0;

while ($i < count($colors)) {
    echo $colors[$i] . " ";
    $i++;
}
```

2. Các cấu trúc điều khiển

- **Cấu trúc với DO .. WHILE:**
  - do-while là một biến thể của vòng lặp while và được sử dụng khi bạn muốn khối mã được thực thi ít nhất một lần trước khi kiểm tra điều kiện. Điểm khác biệt chính so với vòng lặp while thông thường là do-while kiểm tra điều kiện sau khi khối mã đã được thực thi.

```
$i = 1;

do {
    echo $i . " ";
    $i++;
} while ($i <= 5);
```

```
$colors = ["red", "green", "blue"];
$i = 0;

do {
    echo $colors[$i] . " ";
    $i++;
} while ($i < count($colors));
}
```

2. Các cấu trúc điều khiển

- **Cấu trúc với FOR:**
  - for là một cấu trúc điều khiển được sử dụng để thực hiện một khối mã nhiều lần. Vòng lặp for rất hữu ích khi bạn biết trước số lần cần lặp. Nó bao gồm ba phần quan trọng: khởi tạo biến lặp, điều kiện lặp, và cập nhật biến lặp.

```
for ($i = 1; $i <= 10; $i++) {
    echo $i . " ";
}
```

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

for ($i = 0; $i < count($colors); $i++) {
    echo $colors[$i] . " ";
}
```

2. Các cấu trúc điều khiển

- **Cấu trúc với FOREACH:**
  - Bạn có thể duyệt qua các khóa và giá trị của một mảng sử dụng vòng lặp foreach. Điều này rất hữu ích khi bạn cần xử lý cả khóa và giá trị của mỗi phần tử trong mảng..

```php
$person = [
    "name" => "Alice",
    "age" => 25,
    "city" => "New York"
];

foreach ($person as $key => $value) {
    echo $key . ": " . $value . "<br>";
}
```

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

foreach ($colors as $value) {
    echo $value . "<br>";
}
```

- Khi sử dụng foreach với mảng kết hợp, bạn có thể lấy cả khóa và giá trị hoặc chỉ giá trị.
- Nếu bạn thay đổi giá trị của $value trong vòng lặp, điều này không ảnh hưởng đến mảng gốc, trừ khi bạn tham chiếu đến giá trị đó bằng cách sử dụng &$value.
- Vòng lặp foreach là cách tiện lợi và rõ ràng để xử lý từng phần tử của mảng trong PHP.

2. Các cấu trúc điều khiển

- **Sử dụng INCLUDE/REQUIRE:**
  - **include** và **require** là các lệnh được sử dụng để chèn nội dung của một tệp PHP vào một tệp PHP khác trước khi server thực thi nó. Điều này giúp bạn tách biệt mã nguồn thành các phần nhỏ, tái sử dụng mã, và làm cho mã dễ quản lý và bảo trì hơn.
    - Lệnh include sẽ chèn nội dung của một tệp được chỉ định vào tệp hiện tại. Nếu tệp không tồn tại, PHP sẽ phát ra một cảnh báo (warning) nhưng script vẫn tiếp tục thực thi.

```
include 'path/to/file.php';
```

    - Lệnh require hoạt động giống như include, nhưng nếu tệp không tồn tại, nó sẽ phát ra một lỗi nghiêm trọng (fatal error) và dừng việc thực thi script.

```
require 'path/to/file.php';
```

2. Các cấu trúc điều khiển

- **Sử dụng INCLUDE/REQUIRE:**

```
#header.php
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
</head>
<body>
```

```
#footer.php
 <footer>Footer content here</footer>
</body>
</html>
```

```
#menu.php
<nav>
    <ul>
        <li>Home</li>
        <li>About</li>
        <li>Contact</li>
    </ul>
</nav>
```

Ở đây, index.php sử dụng include để thêm các phần header, menu, và footer. Nếu một trong các file này không tồn tại, index.php vẫn sẽ thực thi những phần còn lại sau cảnh báo.

```
#index.php

<?php include 'header.php'; ?>
<?php include 'menu.php'; ?>

<h1>Welcome to My Web Page</h1>
<p>Some content goes here...</p>

<?php include 'footer.php'; ?>
```

- Chọn require khi file bạn chèn là một phần thiết yếu của ứng dụng và không thể chạy đúng nếu file đó thiếu.
- Sử dụng include khi file không quá quan trọng và ứng dụng vẫn có thể chạy mặc dù có thể không hoàn chỉnh.
- Cả hai lệnh này giúp quản lý mã nguồn hiệu quả hơn, đặc biệt trong các dự án lớn.

2. Các cấu trúc điều khiển

- **Sử dụng INCLUDE/REQUIRE:**

include_once và require_once đảm bảo rằng một file chỉ được chèn một lần. Nếu file đã được chèn trước đó, PHP sẽ không chèn nó lại. Điều này hữu ích khi bạn muốn tránh lỗi do định nghĩa hàm hoặc lớp lặp lại.

```
#config.php
<?php
define('DB_HOST', 'localhost');
define('DB_USER', 'username');
define('DB_PASS', 'password');
define('DB_NAME', 'database_name');
```

```
#index.php
<?php
require_once 'config.php';
// Other code
```

```
#other_page.php
<?php
require_once 'config.php';
// Other code
```

Ở đây, dù index.php và another_page.php cùng chèn config.php, nhưng nhờ require_once, config.php chỉ được chèn một lần, tránh được lỗi định nghĩa hằng số lặp lại.

2. Các cấu trúc điều khiển

- **Sử dụng INCLUDE/REQUIRE:**

```
#functions.php
<?php
function doSomething() {
    // function code
}
```

```
#page1.php
<?php
include_once 'functions.php';
doSomething();
```

```
#page2.php
<?php
include_once 'functions.php';
doSomething();
```

include_once và require_once đảm bảo rằng một file chỉ được chèn một lần. Nếu file đã được chèn trước đó, PHP sẽ không chèn nó lại. Điều này hữu ích khi bạn muốn tránh lỗi do định nghĩa hàm hoặc lớp lặp lại.

```
Ở đây, include_once đảm bảo rằng mặc dù
functions.php được yêu cầu ở nhiều nơi, nó chỉ
được chèn một lần, ngăn chặn lỗi do định nghĩa
hàm doSomething lặp lại.
```

2. Các cấu trúc điều khiển

- **Ví dụ:**

```
#products.php
<?php
$products = [
    ['name' => 'Sản phẩm 1', 'price' => '1000'],
    ['name' => 'Sản phẩm 2', 'price' => '2000'],
    ['name' => 'Sản phẩm 3', 'price' => '3000']
];
```

```
project/
│
├── index.php
├── header.php
├── footer.php
└── products.php
```

```
#header.php
<!DOCTYPE html>
<html>
<head>
    <title>Trang Sản Phẩm</title>
    <!-- Các thẻ link hoặc script khác nếu cần -
->
</head>
<body>
    <header>
        <h1>Chào mừng đến với Trang Sản Phẩm của
Chúng Tôi</h1>
    </header>
```

```
#footer.php
 <footer>
        <p>&copy; 2023 Trang Sản Phẩm</p>
    </footer>
</body>
</html>
```

```
#index.php
<?php include 'header.php'; ?>
<?php include 'products.php'; ?>

<main>
    <h2>Danh Sách Sản Phẩm</h2>
    <?php if (empty($products)): ?>
        <p>Không có sản phẩm nào.</p>
    <?php else: ?>
        <ul>
            <?php foreach ($products as $product): ?>
                <li><?= htmlspecialchars($product['name']) ?>:
<?= htmlspecialchars($product['price']) ?> VND</li>
            <?php endforeach; ?>
        </ul>
    <?php endif; ?>
</main>

<?php include 'footer.php'; ?>
```

3. Hàm

- **Hàm** (function) là một khối mã được định nghĩa để thực hiện một nhiệm vụ cụ thể. Hàm có thể được gọi nhiều lần trong chương trình, giúp giảm bớt việc lặp lại mã nguồn và làm cho mã dễ quản lý hơn.

```
#Cú pháp
 function functionName($param1, $param2, ...) {
    // Code to be executed
}
```

```
#Hàm không tham số
function sayHello() {
    echo "Hello!";
}

sayHello(); // Gọi hàm
```

```
#Hàm có tham số
function greet($name) {
    echo "Hello, " . $name . "!";
}

greet("Alice"); // Gọi hàm với tham số "Alice"
```

```
#Hàm trả về giá trị
function add($num1, $num2) {
    return $num1 + $num2;
}

$result = add(5, 10); // $result sẽ có giá trị 15
```

```
#Hàm trả về nhiều giá trị
function calculate($num1, $num2) {
    $sum = $num1 + $num2;
    $product = $num1 * $num2;
    return array($sum, $product);
}

$result = calculate(10, 5);
echo "Tổng: " . $result[0] . "; Tích: " .
$result[1];
```

3. Hàm

- **Phạm vi (scope)** của biến quyết định nơi bạn có thể truy cập và sửa đổi biến đó trong mã của mình. Có hai loại phạm vi chính là phạm vi cục bộ (local) và toàn cục (global).
  - Phạm vi cục bộ (local scope)

```
function testFunction() {
    $localVar = "Hello"; // Biến cục bộ
    echo $localVar;
}

testFunction(); // In ra: Hello
// echo $localVar; // Sẽ gây ra lỗi vì $localVar không
tồn tại bên ngoài hàm
```

Phạm vi toàn cục (global scope)

```
$globalVar = "Hello";

function testGlobal() {
    global $globalVar; // Khai báo sử dụng biến toàn cục
    echo $globalVar;
}

testGlobal(); // In ra: Hello
echo $globalVar; // In ra: Hello
```

  - Sử dụng tử khóa global: Để truy cập một biến toàn cục từ bên trong một hàm, bạn cần sử dụng từ khóa global trước khi sử dụng biến đó.
    - Một cách khác để truy cập biến toàn cục từ bên trong một hàm là sử dụng mảng $GLOBALS. $GLOBALS là một mảng đặc biệt trong PHP mà lưu trữ tất cả các biến toàn cục dưới dạng một mảng.

```
$globalVar = "Hello";

function testGlobals() {
    echo $GLOBALS['globalVar'];
}

testGlobals(); // In ra: Hello
```

3. HÀM

- **Biến toàn cục và biến tĩnh**
  - Biến toàn cục: Biến toàn cục là những biến được khai báo bên ngoài tất cả các hàm. Biến toàn cục có thể truy cập được ở mọi nơi trong chương trình, nhưng để sử dụng chúng trong một hàm, bạn cần khai báo rõ ràng bằng từ khóa global.

```
$globalVar = "Hello World"; // Biến toàn cục

function testGlobal() {
    global $globalVar;
    echo $globalVar; // Truy cập biến toàn cục bên trong hàm
}

testGlobal(); // In ra: Hello World
```

  - Biến tĩnh: Biến tĩnh là những biến được khai báo bên trong một hàm với từ khóa static. Khi một hàm được thực thi, giá trị của biến tĩnh sẽ không bị mất đi khi hàm kết thúc. Biến tĩnh giữ giá trị của nó giữa các lần gọi hàm và chỉ được khởi tạo một lần.

```
function testStatic() {
    static $count = 0; // Biến tĩnh
    $count++;
    echo $count . " ";
}

testStatic(); // In ra: 1
testStatic(); // In ra: 2
testStatic(); // In ra: 3
```

3. HÀM

- **Khai báo kiểu**
  - Từ phiên bản PHP 7.0 trở lên, PHP hỗ trợ khai báo kiểu (type declarations) cho tham số của hàm và giá trị trả về. Khai báo kiểu giúp đảm bảo rằng dữ liệu đầu vào và đầu ra của hàm là đúng kiểu mong muốn, làm tăng tính chắc chắn và giảm lỗi trong chương trình.

```
function sum(int $a, int $b) {
    return $a + $b;
}

echo sum(5, "10"); // 15
```

```
function getArray(): array {
    return ["A", "B", "C"];
}

print_r(getArray()); // In ra: Array ( [0] => A [1] => B [2] => C )
```

  - **Strict Typing**: Mặc định, PHP sử dụng "weak typing" nghĩa là nó tự động chuyển đổi kiểu dữ liệu nếu cần. Tuy nhiên, bạn có thể bật "strict typing" để yêu cầu PHP phải nghiêm ngặt tuân theo kiểu dữ liệu đã khai báo.

```
declare(strict_types=1);

function add(int $a, int $b): int {
    return $a + $b;
}

echo add(5, 10); // 15
// echo add(5, "10"); // Sẽ gây ra lỗi Fatal Error
```

3. HÀM

- **Hàm với tham số có giá trị mặc định**
  - Trong PHP, bạn có thể đặt giá trị mặc định cho tham số của hàm. Điều này có nghĩa là nếu khi gọi hàm, không có giá trị nào được cung cấp cho tham số đó, thì giá trị mặc định sẽ được sử dụng. Điều này giúp hàm của bạn linh hoạt hơn và có thể xử lý trường hợp khi không có đủ thông tin đầu vào..

```
function functionName($param1 = 'defaultValue1', $param2 = 'defaultValue2', ...) {
    // Code to be executed
}
```

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

greet("Alice"); // In ra: Hello, Alice!
greet();        // In ra: Hello, Guest!
```

```
function calculate($num1, $num2 = 10) {
    return $num1 + $num2;
}

echo calculate(5);    // In ra: 15
echo calculate(5, 5); // In ra: 10
```

3. HÀM

- **Hàm với đối số được đặt tên**
  - Từ phiên bản PHP 8.0 trở lên, PHP hỗ trợ tính năng gọi hàm với đối số được đặt tên (named arguments), cho phép bạn truyền giá trị vào hàm dựa trên tên của các tham số thay vì dựa trên vị trí của chúng. Tính năng này giúp mã nguồn dễ đọc hơn và linh hoạt hơn, đặc biệt trong trường hợp hàm có nhiều tham số.

```
function setCookieOptions($name, $value, $expire, $path, $secure, $httponly) {
    // code to set cookie options
}
```

```
setCookieOptions(
    name: 'user',
    value: 'Alice',
    expire: time() + 3600,
    path: '/',
    secure: true,
    httponly: true
);
```

```
function createAccount($username, $password, $email = '', $country = 'US') {
    // code to create account
}

createAccount(
    username: 'alice',
    password: 'secret'
);
```

4. LỚP VÀ ĐỐI TƯỢNG

- **Đối tượng và thuộc tính**
  - Đối tượng (objects) và thuộc tính (properties) là những khái niệm cơ bản trong lập trình hướng đối tượng (OOP). Lập trình hướng đối tượng cho phép bạn tạo ra các cấu trúc dữ liệu phức tạp và tái sử dụng mã nguồn một cách hiệu quả.
  - **Định nghĩa lớp**

```php
class Person {
    public $name;
    public $age;

    function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }

    function greet() {
        return "Hello, my name is " . $this->name;
    }
}
```

  - **Tạo đối tượng**

```php
$alice = new Person("Alice", 25);
echo $alice->greet(); // In ra: Hello, my name is Alice
```

  - **Thuộc tính**
    - Thuộc tính công cộng (public): có thể truy cập từ bên ngoài lớp. Trong ví dụ trên, name và age là công cộng.
    - Thuộc tính bảo vệ (protected): chỉ có thể truy cập bên trong lớp đó và các lớp con.
    - Thuộc tính riêng tư (private): chỉ có thể truy cập bên trong lớp đó.

4. LỚP VÀ ĐỐI TƯỢNG

- **Đối tượng và phương thức**
  - Đối tượng là một thực thể cụ thể của một lớp (class), và phương thức là hành động mà đối tượng có thể thực hiện..
  - **Định nghĩa lớp**

```
class Person {
    public $name;

    function __construct($name) {
        $this->name = $name;
    }

    public function greet() {
        return "Hello, my name is " . $this->name;
    }
}
```

  - **Tạo đối tượng**

```
$alice = new Person("Alice");
echo $alice->greet(); // In ra: Hello, my name is Alice
```

  - **Phương thức**
    - Phương thức công cộng (public methods): có thể được gọi từ bên ngoài lớp. Trong ví dụ trên, greet() là một phương thức công cộng.
    - Phương thức bảo vệ (protected methods): chỉ có thể được gọi bên trong lớp đó và các lớp con của nó.
    - Phương thức riêng tư (private methods): chỉ có thể được gọi bên trong lớp đó.

4. LỚP VÀ ĐỐI TƯỢNG

- **Hàm tạo**
  - Một constructor là một phương thức đặc biệt trong một lớp, được tự động gọi khi một đối tượng của lớp đó được tạo. Constructor thường được sử dụng để khởi tạo các thuộc tính của đối tượng hoặc thực hiện bất kỳ thiết lập nào cần thiết khi đối tượng được tạo.

Khi tạo đối tượng $alice, constructor của lớp Person được gọi với các đối số "Alice" và 25.

```
class Person {
    public $name;
    public $age;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
}
```

```
$alice = new Person("Alice", 25);
echo $alice->name; // In ra: Alice
```

- Không cần phải trả về giá trị: constructor không trả về giá trị. Nó chỉ được sử dụng để khởi tạo đối tượng.
- Một constructor cho mỗi lớp: một lớp chỉ có thể có một constructor. Nếu bạn định nghĩa nhiều hơn một, chỉ constructor cuối cùng được định nghĩa sẽ được sử dụng.
- Gọi constructor của lớp cha: nếu lớp con kế thừa từ một lớp cha có constructor, bạn có thể gọi constructor của lớp cha bằng cách sử dụng parent::__construct().

4. LỚP VÀ ĐỐI TƯỢNG

- **Getter và Setter**
  - getters và setters là các phương thức được sử dụng để truy cập và thiết lập giá trị của các thuộc tính riêng tư hoặc bảo vệ (private/protected) của một lớp. Việc sử dụng getters và setters giúp tăng cường tính bảo mật và kiểm soát truy cập đến dữ liệu của đối tượng..

```php
$person = new Person();
$person->setName("Alice");
$person->setAge(25);

echo $person->getName(); // In ra: Alice
echo $person->getAge();  // In ra: 25
```

```php
class Person {
    private $name;
    private $age;

    public function getName() {
        return $this->name;
    }
    public function setName($name) {
        if (is_string($name) && strlen($name) > 0) {
            $this->name = $name;
        }
    }

    public function getAge() {
        return $this->age;
    }
    public function setAge($age) {
        if (is_int($age) && $age > 0) {
            $this->age = $age;
        }
    }
}
```

4. LỚP VÀ ĐỐI TƯỢNG

- **Mảng trong thuộc tính đối tượng**
  - Sử dụng trong các ứng dụng lớn cần quản lý một tập hợp phức tạp của dữ liệu liên quan đến đối tượng đó.

```
class Person {
    private $name;
    private $hobbies;

    public function __construct($name) {
        $this->name = $name;
        $this->hobbies = array();
    }

    public function addHobby($hobby) {
        $this->hobbies[] = $hobby;
    }

    public function getHobbies() {
        return $this->hobbies;
    }
}
```

```
$person = new Person("Alice");
$person->addHobby("Reading");
$person->addHobby("Cycling");

print_r($person->getHobbies()); // In ra mảng sở thích
```

4. LỚP VÀ ĐỐI TƯỢNG

- **Đối tượng trong thuộc tính đối tượng**
  - Gán một đối tượng làm thuộc tính của một đối tượng khác, tạo ra một cấu trúc phức tạp hơn của dữ liệu.

```
class Address {
    public $street;
    public $city;

    public function __construct($street, $city) {
        $this->street = $street;
        $this->city = $city;
    }
}

class Person {
    public $name;
    public $address; // Address là một đối tượng

    public function __construct($name, Address $address) {
        $this->name = $name;
        $this->address = $address;
    }
}
```

```
$address = new Address("123 Main St", "Anytown");
$person = new Person("Alice", $address);

echo $person->name; // In ra: Alice
echo $person->address->city; // In ra: Anytown
```

Trong ví dụ này, một đối tượng Address được tạo và sau đó được gán làm thuộc tính cho một đối tượng Person.

“Câu hỏi & Thảo luận”

THE END!