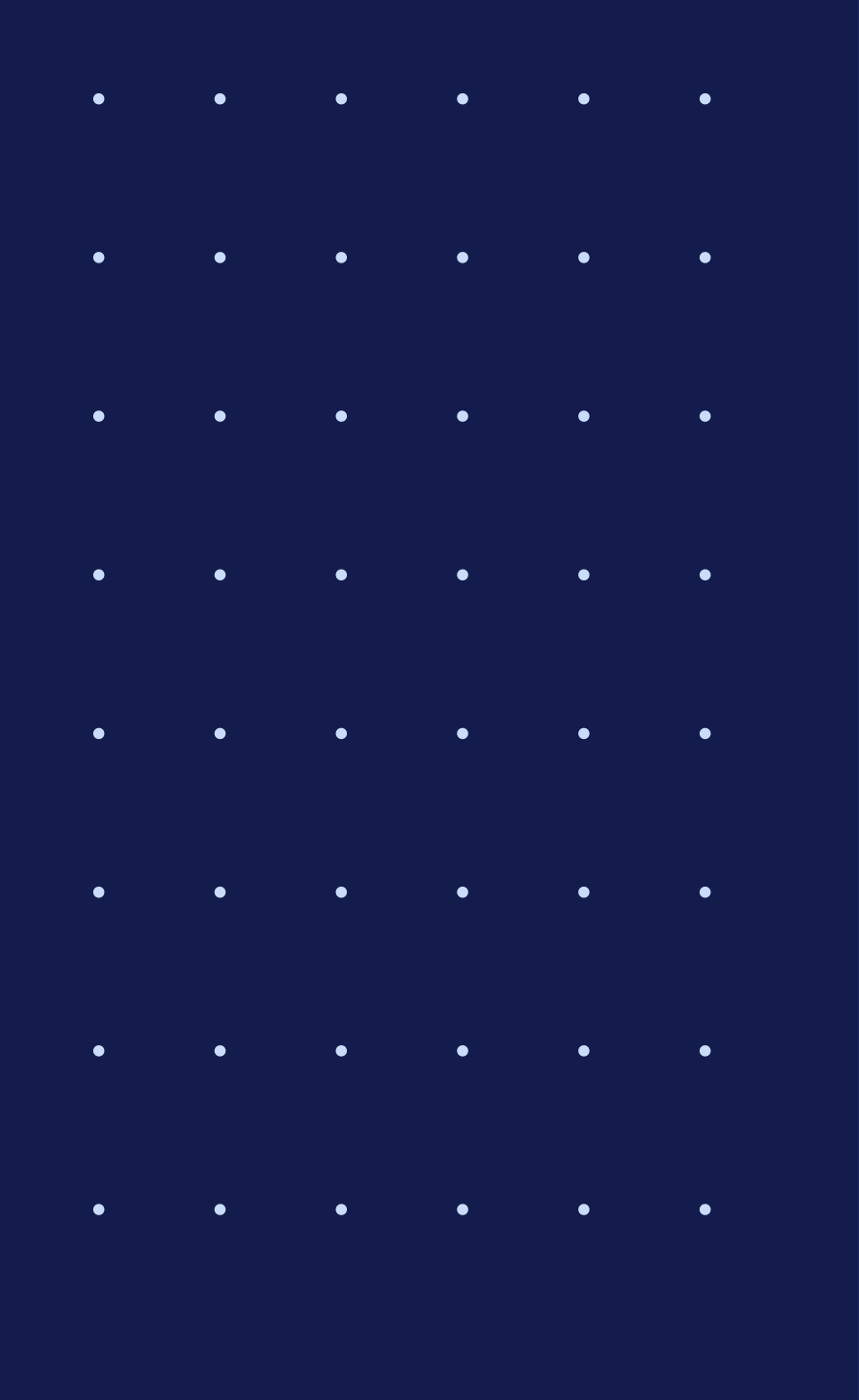




ТЕМА 4

Функції, ООП та ін'єкція залежностей у PHP

Функціональне програмування · класи та об'єкти · модифікатори доступу · простори імен · Dependency Injection

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 4 з 18

10 питань сьогоднішньої теми

01

**Функції:
оголошення та
виклик**

02

**Параметри
функцій**

03

**Типізація
параметрів і return**

04

**Анонімні функції та
замикання**

05

**Функції як дані. ФП
в PHP**

06

**ООП: класи та
об'єкти**

07

**Модифікатори
доступу**

08

**Успадкування,
інтерфейси,
абстракції**

09

Простори імен

10

**Ін'єкція
залежностей (DI)**

Оголошення та виклик

```
<?php
function greet($name) {
    return "Привіт, $name!";
}

echo greet("Оксана"); // Привіт, Оксана!
echo greet("Іван");   // Привіт, Іван!
```

Навіщо потрібні функції

- Уникнення дублювання коду (DRY)
- Розбиття задачі на менші частини
- Повторне використання логіки

За замовчуванням, іменовані, варіативні

За замовчуванням

```
function greet($name, $g = "Привіт")
```

Іменовані (PHP 8.0+)

```
createUser(name: "Тарас", city: "Київ")
```

Варіативні (...)

```
function sum(...$numbers)
```

Типи аргументів і значення, що повертається

```
function addTax(float $price, float $rate = 0.2): float {  
    return $price * (1 + $rate);  
}  
  
echo addTax(100);           // 120  
  
// Тип, що допускає null – префікс ?  
function findUser(?int $id): ?string {  
    return $id === null ? null : "Користувач #$id";  
}
```

declare(strict_types=1) вимикає автоматичне приведення типів аргументів — без нього `addTax("100")` спрацює, а з ним викине `TypeError`.

Захоплення змінних через use

```
$tax = 0.2;
$addTax = function($price) use ($tax) {
    return $price * (1 + $tax);
};
echo $addTax(100);    // 120

$counter = 0;
$increment = function() use (&$counter) {
    $counter++;    // за посиланням – змінює оригінал
};
$increment();
$increment();
echo $counter;    // 2
```

function() {} vs fn() =>

Анонімна функція

Потребує явний use() для доступу до зовнішніх змінних

```
$tax = 0.2;  
$f = function($p) use ($tax) {  
    return $p * (1 + $tax);  
};
```

Стрілкова функція (PHP 7.4+)

Автоматично захоплює змінні за значенням, без use

```
$tax = 0.2;  
$f = fn($p) => $p * (1 + $tax);  
  
echo $f(100); // 120
```

Callable та функції вищого порядку

```
// Функція, що повертає функцію (вищого порядку)
function multiplier($factor) {
    return fn($x) => $x * $factor;
}

$triple = multiplier(3);
echo $triple(7);    // 21

// Рядок з іменем функції теж callable
$doubled = array_map('multiplier', [1, 2, 3]);
```

PHP — мультипарадигмена мова: процедурний, ООП і функціональний стилі можна поєднувати в одному застосунку.

Клас — шаблон, об'єкт — екземпляр

```
class Product {  
    public string $name;  
    public float $price;  
  
    public function __construct(string $name, float $price) {  
        $this->name = $name;  
        $this->price = $price;  
    }  
  
    public function getPriceWithTax(float $rate = 0.2): float {  
        return $this->price * (1 + $rate);  
    }  
}  
  
$laptop = new Product("Ноутбук", 25000);  
echo $laptop->getPriceWithTax();    // 30000
```

public, protected, private

public

Доступний звідусіль

protected

Клас + нащадки

private

Лише сам клас

```
class BankAccount {  
    private float $balance = 0;  
  
    public function deposit(float $amount): void {  
        if ($amount > 0) $this->balance += $amount;  
    }  
    public function getBalance(): float { return $this->balance; }  
}  
// $account->balance; // Помилка! Властивість private
```

extends: перевизначення методів

```
class Animal {  
    public function __construct(protected string $name) {}  
  
    public function makeSound(): string {  
        return "$this->name видає звук";  
    }  
}  
  
class Dog extends Animal {  
    public function makeSound(): string {  
        return "$this->name гавкає";    // перевизначення (override)  
    }  
}  
  
$dog = new Dog("Пекс");  
echo $dog->makeSound();    // Пекс гавкає
```

Контракт vs спільна реалізація

interface

Лише контракт — перелік методів без реалізації

```
interface PaymentMethod {  
    public function pay(  
        float $amount  
    ): bool;  
}
```

abstract class

Спільна логіка + методи, які нащадок мусить реалізувати

```
abstract class Shape {  
    abstract public function  
        getArea(): float;  
    public function describe() {...}  
}
```

namespace та use: уникнення конфліктів імен

```
// файл src/Payment/CardPayment.php
namespace App\Payment;

class CardPayment {
    public function pay(float $amount): void {
        echo "Оплата $amount грн";
    }
}

// файл src/index.php
use App\Payment\CardPayment;

$payment = new CardPayment();
$payment->pay(500);
```

Проблема: жорстко "зашита" залежність

```
class OrderService {  
    private Database $db;  
  
    public function __construct() {  
        $this->db = new MySQLDatabase(); // жорстко "зашита"  
    }  
}
```

Tightly coupled: клас сам створює залежність усередині — його неможливо протестувати без реальної бази даних і неможливо підмінити реалізацію.

Рішення: конструкторна ін'єкція

```
interface Database {  
    public function query(string $sql): array;  
}  
  
class MySQLDatabase implements Database { /* ... */ }  
  
class OrderService {  
    // Конструкторна ін'єкція залежності  
    public function __construct(private Database $db) {}  
  
    public function getOrders(): array {  
        return $this->db->query("SELECT * FROM orders");  
    }  
}  
  
$service = new OrderService(new MySQLDatabase());
```

Підсумок теми 4

- Функції — об'єкти першого класу: їх можна зберігати, передавати, повертати
- Стрілкові функції `fn()` автоматично захоплюють зовнішні змінні
- Клас — шаблон, об'єкт — його екземпляр із власним станом
- `public` / `protected` / `private` керують інкапсуляцією даних класу
- Інтерфейс — контракт; абстрактний клас — спільна логіка + контракт
- DI усуває жорстку зв'язаність: залежність передається ззовні