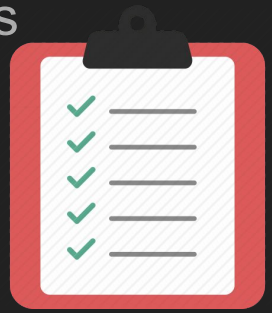
# POO

## Programmation Orienté Objet


**Rachid EDJEKOUANE (edjek@hotmail.fr)**

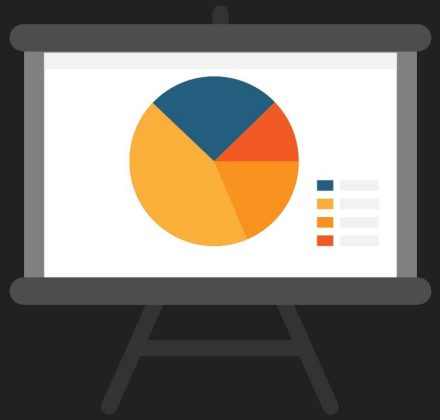
# Prérequis

- Notions d’Algorithmique
- Base de la programmation
- Familiarisé avec PHP
- Compréhension des concepts de base de données
- Bases HTML et CSS
- Ponctualité



php POO

# Présentation



# Présentation

Programmation Orientée Objet (POO) en PHP ! Cette formation a pour objectif de vous initier aux principes fondamentaux de la POO et de vous fournir les connaissances et les compétences nécessaires pour développer des applications web robustes et modulaires en utilisant PHP.

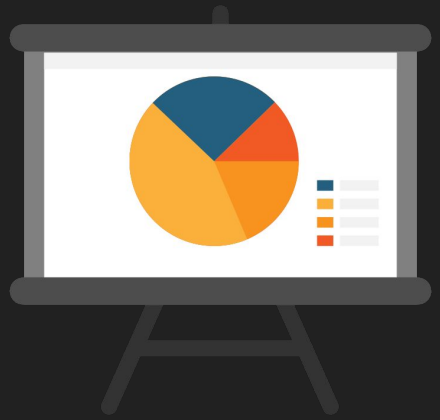
Au cours de cette formation de 5 jours, nous allons explorer les concepts clés de la POO et les appliquer dans le contexte de PHP. Nous aborderons des sujets tels que les classes, les objets, l'encapsulation, l'héritage, le polymorphisme et bien plus encore.

# Objectifs

- Maîtriser les concepts de classe, d'objet, de propriétés et de méthodes en PHP.
- Apprendre à encapsuler le code pour le rendre plus modulaire, réutilisable et sécurisé.
- Explorer l'héritage et le polymorphisme pour favoriser la réutilisation du code et la flexibilité.
- Savoir modéliser et concevoir des classes pour résoudre des problèmes spécifiques.
- Acquérir les bonnes pratiques de développement orienté objet en PHP.

php POO

# Introduction



# Programmation Orienté Objet


La Programmation Orientée Objet est un paradigme de développement logiciel qui repose sur le concept d'objets et de classes.

La POO permet de structurer et d'organiser le code de manière modulaire, en regroupant les données et les fonctionnalités associées dans des entités autonomes appelées objets.

# Programmation Orienté Objet

Dans la POO, un objet est une instance d'une classe.

Une classe définit les caractéristiques (appelées propriétés ou attributs) et les comportements (appelés méthodes) d'un objet.

Les propriétés représentent les données associées à un objet, tandis que les méthodes représentent les actions que l'objet peut effectuer.

# Découverte

# POO : les classes

```php
class Voiture {
    public $marque;
    public $couleur;

    public function demarrer() {
        // Logique pour démarrer la voiture
    }
}
```

# POO : instance de class

Une classe est un modèle ou un plan qui définit la structure et le comportement d'un ensemble d'objets.

Elle regroupe les propriétés et les méthodes qui sont communes à tous les objets de cette classe.

```php
$mini = new Voiture();
$mini->marque = 'bmw';
```

# POO : les objets

Un objet est une instance spécifique d'une classe.

Il est créé en utilisant le mot-clé "new" suivi du nom de la classe.
Chaque objet possède ses propres valeurs de propriétés, mais partage les mêmes méthodes définies dans la classe.

```php
$nomObjet = new NomClass();
```

# POO : encapsulation et visibilité

L'encapsulation est le principe de cacher les détails d'implémentation d'une classe et de n'exposer que les fonctionnalités nécessaires.

En PHP, vous pouvez spécifier la visibilité des propriétés et des méthodes à l'aide des mots-clés "public", "protected" ou "private".

# POO : visibilité

```php
class Personne {
    public string $nom; // Propriété publique
    private int $age; // Propriété privée
    protected string $nationalite // Propriété protégée

// Méthode publique pour accéder à une propriété privée
    public function getAge(): string {
        return $this->age;  }
}
```

# POO : phpdoc

PHPDoc est une convention de documentation utilisée pour annoter le code PHP avec des commentaires spécifiques afin de générer une documentation claire et complète.

Cela permet aux développeurs et aux outils d'analyse statique de mieux comprendre la structure et l'utilisation du code.

# POO : phpdoc

```php
/**
 * Description de la fonction.
 *
 * @param string $param1 Description du paramètre 1.
 * @param int $param2 Description du paramètre 2.
 * @return bool Description du type de retour.
 */
function myFunction(string $param1, int $param2): bool {
    // Code de la fonction
}
```

# POO : méthodes magiques

Les méthodes magiques sont des méthodes spéciales définies dans une classe qui sont appelées automatiquement dans certaines situations.

Par exemple, la méthode __construct() est appelée lors de l'instanciation d'un objet.

# POO : méthodes magiques

```php
class MaClasse {
    private $nom;
    private $age;

    public function __construct($nom, $age) {
        $this->nom = $nom;
        $this->age = $age;
    }
}  // Création d'une instance de MaClasse en passant des valeurs au constructeur
$objet = new MaClasse("John Doe", 30);
```

# POO : method static

En PHP, une classe peut contenir des membres (propriétés et méthodes) statiques.

Les membres statiques sont des éléments de classe qui sont partagés entre toutes les instances de cette classe. Ils sont liés à la classe elle-même plutôt qu'à une instance spécifique de cette classe.

# POO : method static (en pratique)

```php
class Mathematique
{
    public function addition($a, $b)
    {
        return $a + $b;
    }
    public static function division($a, $b)
    {
        return $a / $b;
    }
}
echo Mathematique::division(12, 78);
```

# POO : héritage

L'héritage permet de créer une classe dérivée (ou enfant) à partir d'une classe existante (ou parent).

La classe dérivée hérite des propriétés et des méthodes de la classe parent et peut les étendre ou les modifier selon ses besoins.

En PHP, utilisez le mot-clé "extends" pour définir une relation d'héritage.

# POO : héritage (en pratique)

```php
class Animal {
    protected function faireDuBruit() {
        echo "Je suis un animal !";
    }
}

class Chien extends Animal {
    // La classe Chien hérite de la méthode faireDuBruit() de la
classe Animal
}

$chien = new Chien();
$chien->faireDuBruit(); // Affiche "Je suis un animal !"
```

# POO : polymorphisme

Le polymorphisme permet à des objets de différentes classes de répondre de manière différente à une même méthode.

En PHP, le polymorphisme peut être réalisé en utilisant l'héritage et en redéfinissant des méthodes dans les classes dérivées.

# POO : polymorphisme (en pratique)

```php
class Forme {
    public function dessiner() {
        echo "Je suis une forme générique.";
    }
}

class Cercle extends Forme {
    public function dessiner() {
        echo "Je suis un cercle et " .  parent::dessiner();
    }
}
```

# POO : classe abstraite

En PHP, une classe abstraite est une classe qui ne peut pas être instanciée directement, mais qui sert de modèle pour les classes dérivées.

Une classe abstraite peut contenir des méthodes abstraites, qui sont des méthodes déclarées sans implémentation dans la classe abstraite. Les classes dérivées de la classe abstraite doivent implémenter ces méthodes abstraites.

# POO : class abstract (en pratique)

```php
abstract class Forme {
public abstract function dessiner();
}

class Cercle extends Forme {
    public function dessiner() {
        echo "Je suis un cercle.";
    }
}
```

# POO : interface

En PHP, une interface est un concept clé de la programmation orientée objet (POO) qui permet de définir un contrat ou un ensemble de méthodes que les classes doivent implémenter.

Une interface définit la signature (le nom et les paramètres) des méthodes, mais n'implémente pas leur comportement.

Les classes qui implémentent une interface doivent fournir une implémentation concrète (le corps de la méthode) pour toutes les méthodes définies dans l'interface.

# POO : interface (en pratique)

```php
interface Animal{
    public function eat();
}

class Dog implements Animal{
    public function eat() {
        echo "Le chien mange !";
    }
}
```

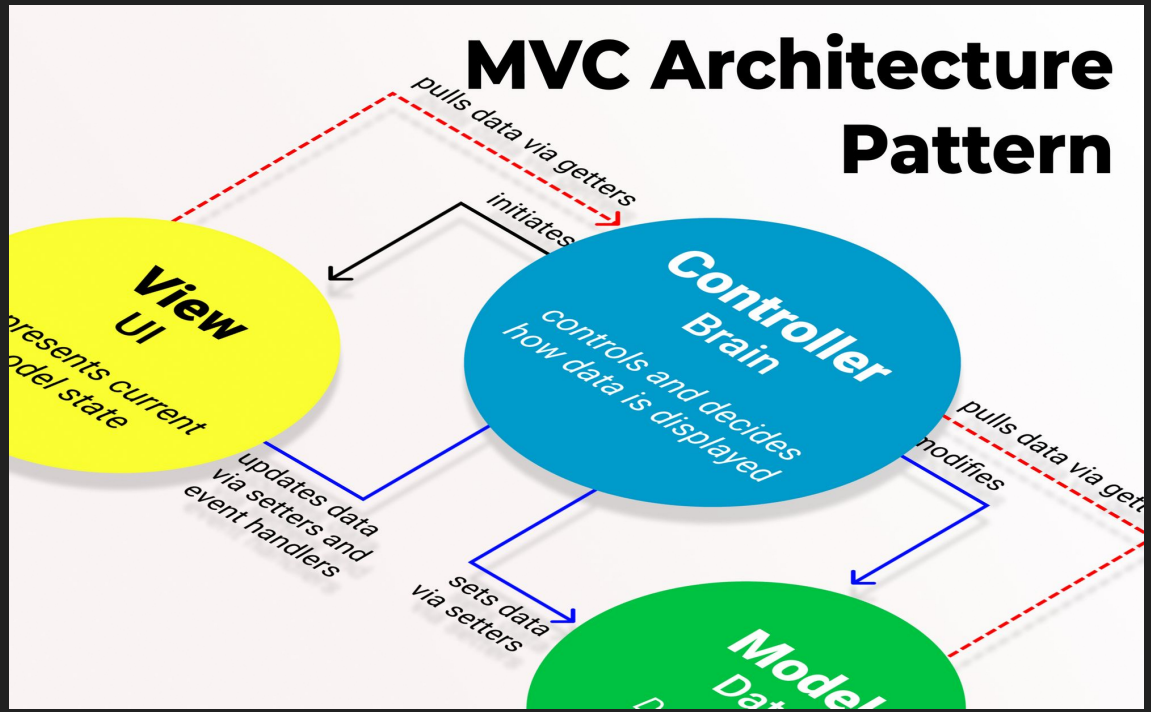
# Architecture logicielle

# POO : architecture logicielle

L'architecture logicielle en PHP fait référence à la structure organisationnelle et à la conception globale d'une application PHP.

Elle définit la manière dont les composants de l'application sont organisés, interagissent entre eux et répondent aux besoins fonctionnels et non fonctionnels de l'application.

# POO : mvc


MVC Architecture Pattern
pulls data via getters
initiates
View
UI
resents current
odel state
Controller
Brain
controls and decides
how data is displayed
updates data
via setters and
event handlers
sets data
via setters
pulls data via get
nodifies
Model

# POO : mvc

Le modèle-vue-contrôleur (MVC) est un motif d'architecture logicielle couramment utilisé pour développer des applications web.

Il permet de séparer les préoccupations liées aux données, à la logique métier et à la présentation dans des composants distincts pour une meilleure organisation et une plus grande maintenabilité du code.

# POO : modèle

Modèle (Model) : Le modèle représente les données de l'application et contient la logique métier associée.

Il gère l'accès aux données, effectue des opérations de lecture/écriture et implémente les règles métier. Le modèle ne connaît pas l'existence de la vue ou du contrôleur.

# POO : vue

Vue (View) : La vue est responsable de l'affichage des données et de l'interaction avec l'utilisateur.

Elle reçoit les données du modèle et les présente de manière appropriée. La vue ne contient pas de logique métier significative, son rôle principal est de rendre les données pour une visualisation.

# POO : contrôleur

Contrôleur (Controller) : Le contrôleur gère les actions de l'utilisateur et les interactions avec le modèle et la vue.

Il reçoit les requêtes de l'utilisateur, effectue les opérations appropriées sur le modèle et détermine quelle vue doit être affichée en fonction de l'état de l'application. Le contrôleur est responsable de la coordination entre le modèle et la vue.

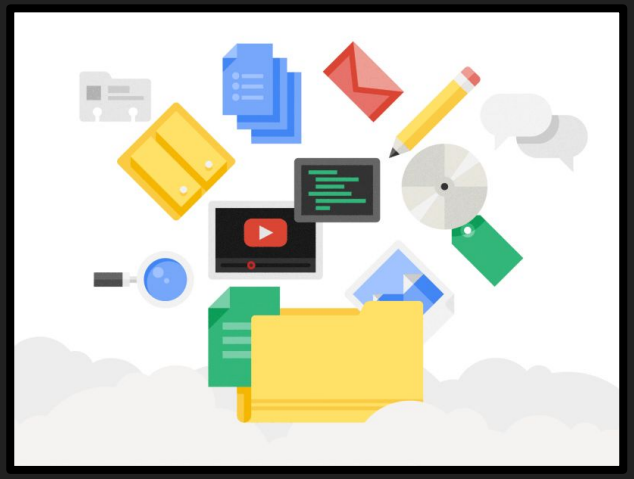
# POO : mise en oeuvre

Le modèle MVC favorise une séparation claire des responsabilités et permet une évolutivité et une maintenabilité accrues du code.

Il facilite également le travail en équipe en permettant une division des tâches entre les développeurs travaillant sur différents composants.

php POO

# Namespace



# POO : espaces de noms

Les espaces de noms (namespace) en PHP sont utilisés pour organiser et regrouper des classes, des fonctions et d'autres éléments dans un espace de noms spécifique.

Ils permettent d'éviter les conflits de noms et de fournir une structure plus claire et plus cohérente pour les classes et les fichiers dans un projet PHP.

# POO : interface (en pratique)

```php
<?php

namespace App\Entity;

class Animal
{
    private string $name;
}
```

# Autoloader

# POO : autoload

Un autoloader en PHP est un mécanisme qui permet de charger automatiquement les classes lorsqu'elles sont utilisées pour la première fois, sans avoir besoin d'inclure manuellement les fichiers correspondants à chaque classe.

# Router

# POO : router

Un routeur (ou router) en PHP est un mécanisme qui permet de diriger les requêtes HTTP vers les bonnes actions ou pages dans une application web.

Il permet de définir des règles de correspondance entre les URL et les actions à exécuter.

# Plus loin…

# POO :  design patterns

Design patterns (Modèles de conception) :

Les design patterns sont des solutions éprouvées aux problèmes de conception courants. Ils fournissent des modèles reconnus pour structurer et organiser votre code de manière optimale.

Certains patterns couramment utilisés en POO incluent le Singleton, le Factory, l'Observer, etc…

# PHP : Lectures complémentaires

Consultez certains des liens pour mieux comprendre le fonctionnement de la POO:

PHP Poo : Documentation officielle

Programmez en POO avec PHP : Cours OpenClassrooms

Programmation Orienté Objet : Tuto vidéo de Grafikart

```
$message->get( 'MERCI');
```