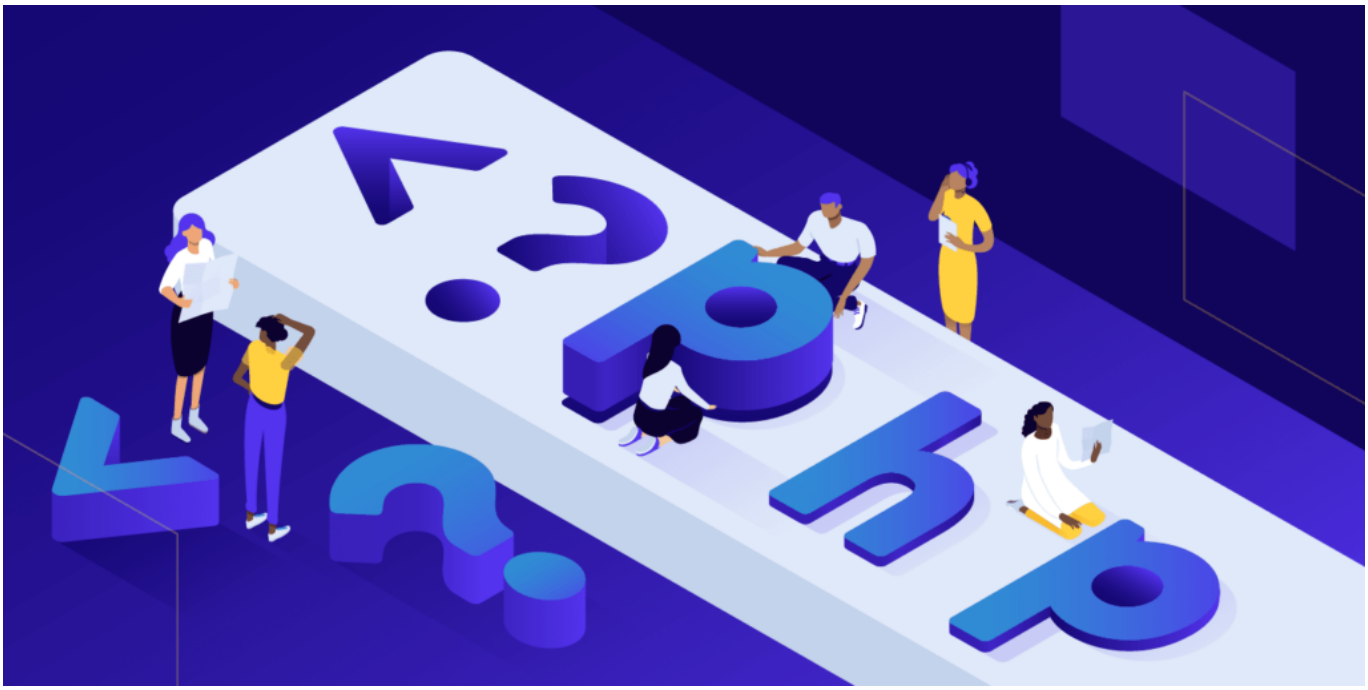
APPRENDRE À CODER AVEC PHP



Introduction

PHP, qui signifie `PHP Hypertext Preprocessor`, est un langage de `script` côté serveur principalement utilisé pour le développement web. Depuis sa création en 1994 par `Rasmus Lerdorf`, il a subi de

nombreuses évolutions et est devenu l'un des langages de programmation web les plus populaires au monde.

1.1. Historique de PHP

Au départ, `PHP` était simplement un ensemble de scripts personnels écrits par Rasmus Lerdorf pour gérer sa page d'accueil. Il a ensuite été repensé et réécrit par `Zeev Suraski` et `Andi Gutmans` pour devenir un moteur de script complet, le `Zend Engine`. Depuis lors, `PHP` a connu plusieurs versions majeures, chacune apportant son lot d'améliorations et de nouvelles fonctionnalités. Aujourd'hui, il alimente d'importants sites web comme `Facebook`, `Wikipedia` et `WordPress`.

1.2. Avantages et inconvénients de PHP

PHP possède de nombreux avantages qui expliquent sa popularité :

1. **Facilité d'utilisation** : Grâce à sa syntaxe simple, PHP est facile à apprendre pour les débutants.
2. **Flexibilité** : Il peut être intégré directement dans le code HTML.
3. **Open Source** : PHP est gratuit et bénéficie d'une grande communauté de développeurs.
4. **Portabilité** : Compatible avec la plupart des serveurs web et des systèmes d'exploitation.
5. **Performance** : Avec un bon codage et une bonne configuration, PHP peut offrir de hautes performances.

Installation et configuration

PHP est soutenu par un grand nombre de systèmes d'exploitation et de serveurs web. Dans cette section, nous aborderons les étapes d'installation de PHP sur Windows et Linux, ainsi que la configuration de base pour le serveur Apache.

2.1. Exigences système

Avant de commencer l'installation de PHP, assurez-vous que votre système répond aux exigences minimales :

- Processeur : `1 GHz` ou supérieur
- Mémoire : `512 Mo` de `RAM` (`1 Go` recommandé pour une performance optimale)
- Disque dur : `200 Mo` d'espace libre (l'espace requis peut varier en fonction des extensions installées)

2.2. Installation sur Windows

1. Visitez le site officiel de PHP à l'adresse php.net et téléchargez la version Windows de PHP.
2. Décompressez l'archive dans le répertoire de votre choix, par exemple `C:\php\`.
3. Ajoutez `C:\php\` à votre variable d'environnement PATH.
4. Copiez `php.ini-development` vers `php.ini` et effectuez les configurations nécessaires selon vos besoins.

2.3. Installation sur Linux

La plupart des distributions Linux offrent un package PHP disponible dans leur gestionnaire de paquets. Voici comment installer PHP sur Ubuntu:

```
sudo apt update
sudo apt install php
```

Pour d'autres distributions, consultez la documentation appropriée ou utilisez le gestionnaire de paquets spécifique à votre distribution.

2.4. Configuration du serveur Apache

1. Assurez-vous que le module PHP est activé dans Apache. Dans le fichier `httpd.conf`, vérifiez la présence de la ligne: `LoadModule php_module modules/mod_php.so`
2. Dites à `Apache` de traiter les fichiers `.php` avec `PHP`. Ajoutez la ligne suivante :

   ```
   AddType application/x-httpd-php .php
   ```

3. Indiquez à Apache où se trouve le fichier php.ini en ajoutant la ligne suivante :

   ```
   PHPIniDir "C:/php"
   ```

(Adaptez le chemin en fonction de l'emplacement de votre installation PHP sur Windows. Sur Linux, cette étape peut ne pas être nécessaire.)

4. Redémarrez le serveur Apache pour que les modifications prennent effet.

Une fois ces étapes terminées, votre serveur Apache devrait être configuré pour exécuter des scripts PHP!

Notez que les instructions ci-dessus sont générales et peuvent varier en fonction de la version de PHP, du système d'exploitation ou du serveur web que vous utilisez. Consultez toujours la documentation officielle pour obtenir des instructions spécifiques à votre configuration.

2.5. Les serveurs web alternatifs

Outre l'installation manuelle de PHP et d'un serveur web comme Apache ou Nginx, il existe des solutions prêtes à l'emploi qui regroupent généralement PHP, un serveur web et une base de données, simplifiant considérablement la mise en place d'un environnement de développement local pour PHP. Voici quelques-unes des plus populaires :

1. **WampServer (Windows)** :
   - **Description** : WampServer est une plateforme Windows qui permet de développer des applications web dynamiques avec Apache2, PHP et MySQL.
   - **Avantages** : Installation facile, interface utilisateur graphique pour gérer les services et les configurations, possibilité de basculer entre différentes versions de PHP.
   - **Lien** : Site officiel de WampServer
2. **MAMP (Macintosh, Windows)** :

- **Description** : MAMP est une solution gratuite, locale de serveur qui peut être installée sur macOS et Windows avec quelques clics.
- **Avantages** : Facilité d'utilisation, versions Pro disponibles avec des fonctionnalités supplémentaires, interface utilisateur graphique.
- **Lien** : Site officiel de MAMP

3. **XAMPP (Cross-platform)** :

   - **Description** : XAMPP est une distribution Apache facile à installer qui contient MariaDB, PHP et Perl.
   - **Avantages** : Multi-plateforme (Windows, Linux, macOS), panneau de contrôle pour gérer les services, nombreux modules et composants inclus.
   - **Lien** : Site officiel de XAMPP

4. **Laragon (Windows)** :

   - **Description** : Laragon est un environnement de développement rapide et riche en fonctionnalités pour Windows.
   - **Avantages** : Mise en route rapide, légèreté, isolation des projets, excellent pour développer avec Laravel.
   - **Lien** : Site officiel de Laragon

Ces solutions sont particulièrement pratiques pour les développeurs qui souhaitent éviter la configuration manuelle des différents composants nécessaires à un environnement de développement PHP complet. Elles permettent une mise en place rapide et sont souvent accompagnées d'outils pour faciliter la gestion des services et des configurations.

Les bases de PHP

PHP (PHP: Hypertext Preprocessor) est un langage de script côté serveur conçu pour la création de sites web dynamiques. Il est intégré à de nombreux serveurs web et est utilisé par des millions de sites web à travers le monde. Comprendre les bases de PHP est essentiel pour tout développeur web qui souhaite créer des applications interactives et dynamiques.

3.1. Structure d'un script PHP

Un script PHP est généralement inséré dans un fichier HTML et est encadré par les balises `<?php` et `?>`. Tout le code entre ces balises est interprété comme du PHP.

```
<!DOCTYPE html>
<html>
<head>
    <title>Titre de la page</title>
</head>
<body>
    <h1>Mon premier script PHP</h1>
    <?php
        echo "Bonjour le monde!";
    ?>
```

```
</body>
</html>
```

Dans ce code, on remarque la présence de la fonction `echo` qui permet d'afficher du texte dans le navigateur. On peut également utiliser la balise `<?=` pour afficher du texte sans utiliser la fonction `echo`.

Il est aussi possible d'utiliser la balise `<?php` sans les balises de fermeture `?>` Pourquoi ?

Parce que les balises de fermeture `?>` peuvent parfois être source d'erreurs. En effet, si un espace ou un saut de ligne est ajouté après la balise de fermeture, cela peut provoquer des erreurs dans le code. Pour éviter ce problème, il est recommandé d'utiliser uniquement la balise d'ouverture `<?php` lorsque vous écrivez uniquement du code PHP.

ou la syntaxe alternative :

```
<!DOCTYPE html>
<html>
<head>
    <title>Titre de la page</title>
</head>
<body>
    <h1>Mon premier script PHP</h1>
    <?= "Hello World ?">
</body>
</html>
```

Cette syntaxe alternative est activée par défaut dans les versions de PHP supérieures à 5.4.0. Elle est souvent utilisée pour insérer des variables dans le code HTML. Elle remplace la balise `<?php echo` par `<?=`, ce qui permet de réduire la quantité de code à écrire.

3.2. Les commentaires

Les commentaires sont des lignes de code qui ne sont pas exécutées par le serveur web. Ils sont utilisés pour ajouter des notes et des explications au code et pour le rendre plus facile à comprendre. Les commentaires peuvent être ajoutés en utilisant `//` pour les commentaires sur une seule ligne ou `/* */` pour les commentaires sur plusieurs lignes.

Il existe 3 types de commentaires :

- **Commentaires sur une seule ligne** : `// Commentaire sur une seule ligne` ou `# Commentaire sur une seule ligne`
- **Commentaires sur plusieurs lignes** : `/* Commentaire sur plusieurs lignes */`
- **Commentaires de documentation** : `/** Commentaire de documentation */`

```
<?php
    // Ceci est un commentaire sur une seule ligne

    # Ceci est un commentaire sur une seule ligne
```

```
    /*
        Ceci est un commentaire
        sur plusieurs lignes
    */

    /**
     * Ceci est un commentaire de documentation
     */
?>
```

3.3. Variables et types de données

En PHP, une variable commence par le signe $, suivi du nom de la variable. PHP est un langage à typage dynamique, c'est-à-dire, que vous n'avez pas besoin de déclarer le type d'une variable lors de sa création.

En PHP, il existe 8 types de données :

- **String** : Une chaîne de caractères
- **Integer** : Un nombre entier
- **Float** : Un nombre décimal
- **Boolean** : Une valeur booléenne (true ou false)
- **Array** : Un tableau
- **Object** : Un objet
- **NULL** : Une variable sans valeur

```
<?php
  $nom = "John";                                    // Ceci est une
chaîne de caractères
  $age = 25;                                        // Ceci est un nombre
entier
  $taille = 1.75;                                   // Ceci est un nombre
décimal
  $est_majeur = true;                               // Ceci est une
valeur booléenne true/false
  $hobbies = array("Football", "Cinéma", "Musique"); // Ceci est un
tableau
  $skills = ["HTML", "CSS", "JavaScript"];          // Ceci est un
tableau
  $personne = new Personne();                       // Ceci est un objet
  $personne = null;                                 // Ceci est une
variable nulle
?>
```

Certaines règles doivent être respectées lors de la création de variables :

- Le nom d'une variable doit commencer par une lettre ou un tiret bas (underscore). : $_nom, $nom
- Le nom d'une variable ne peut pas commencer par un chiffre ou un tiret : $1nom, $-nom

- Le nom d'une variable ne peut contenir que des caractères alphanumériques et des tirets bas : `$nom`, `$nom1`, `$nom_1`
- Le nom d'une variable est sensible à la casse : `$nom` et `$Nom` sont deux variables différentes
- Le nom d'une variable ne peut pas contenir d'espaces : `$nom de famille`
- Le nom d'une variable ne peut pas contenir de caractères spéciaux : `$nom@`, `$nom#`
- Le nom d'une variable ne peut pas être un mot réservé : `$echo`, `$if`, `$while`
- Le nom d'une variable ne peut pas contenir d'accents : `$prénom`

3.4. Opérateurs

PHP propose une variété d'opérateurs pour effectuer des opérations sur les variables :

- Arithmétiques : `+`, `-`, `*`, `/`, `%`
- Comparaison : `==`, `===`, `!=`, `!==`, `<`, `>`, `<=`, `>=`
- Logiques : `&&`, `||`, `!`
- Affectation : `=`, `+=`, `-=`, `*=`, `/=`, `%=`
- Incrémentation : `++`, `--`
- Concaténation : `.`
- Ternaire : `? :`
- Tableau : `[]`
- Type : `instanceof`
- Exponentiation : `**`
- Opérateur de fusion null : `??`
- Opérateur de coalescence null : `??=`
- Opérateur de navigation null : `?->`

Expliquons-les en détail :

Arithmétiques : Ces opérateurs sont utilisés pour effectuer des opérations arithmétiques.

- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` : division
- `%` : modulo (reste de la division)

```php
<?php
    $a = 10;
    $b = 5;
    echo $a + $b; // Affiche 15
    echo $a - $b; // Affiche 5
    echo $a * $b; // Affiche 50
    echo $a / $b; // Affiche 2
    echo $a % $b; // Affiche 0
?>
```

Comparaison : Ces opérateurs sont utilisés pour comparer deux valeurs.

- `==` : égal à
- `===` : identique (égal et du même type)
- `!=` : différent de
- `!==` : non identique
- `<` : inférieur à
- `>` : supérieur à
- `<=` : inférieur ou égal à
- `>=` : supérieur ou égal à

```php
<?php
    $a = 10;
    $b = 5;
    echo $a == $b;  // Affiche false
    echo $a === $b; // Affiche false
    echo $a != $b;  // Affiche true
    echo $a !== $b; // Affiche true
    echo $a < $b;   // Affiche false
    echo $a > $b;   // Affiche true
    echo $a <= $b;  // Affiche false
    echo $a >= $b;  // Affiche true
?>
```

Logiques : Ces opérateurs sont utilisés pour effectuer des opérations logiques.

- `&&` : ET logique
- `||` : OU logique
- `!` : NOT logique

```php
<?php
    $a = 10;
    $b = 5;
    echo $a > 5 && $b < 10; // Affiche true
    echo $a > 5 || $b < 10; // Affiche true
    echo !($a > 5);         // Affiche false
?>
```

Affectation : Ces opérateurs sont utilisés pour attribuer des valeurs.

- `=` : affectation
- `+=`, `-=`, `*=`, `/=`, `%=` : affectation avec opération

```php
<?php
    $a = 10;
    $b = 5;
    $a += $b; // équivaut à $a = $a + $b;
    $a -= $b; // équivaut à $a = $a - $b;
    $a *= $b; // équivaut à $a = $a * $b;
```

```
    $a /= $b; // équivaut à $a = $a / $b
    $a %= $b; // équivaut à $a = $a % $b;
?>
```

Incrémentation :

- `++` : incrémente de 1
- `--` : décrémente de 1

```
<?php
    $a = 10;
    $a++; // équivaut à $a = $a + 1;
    $a--; // équivaut à $a = $a - 1;
?>
```

Concaténation :

- `.` : concatène deux chaînes de caractères

```
<?php
    $a = "Hello";
    $b = "World";
    echo $a . " " . $b; // Affiche Hello World
?>
```

Ternaire :

- `? :` : opérateur ternaire (condition ? valeur_si_vrai : valeur_si_faux)
- `??` : opérateur de fusion null (valeur1 ?? valeur2)
- `??=` : opérateur de coalescence null (variable ??= valeur)
- `?->` : opérateur de navigation null (variable?->propriété)

```
<?php
    $a = 10;
    $b = 5;
    echo $a > $b ? "a est supérieur à b" : "a est inférieur à b"; //
Affiche a est supérieur à b
    echo $a ?? $b;  // Affiche 10
    echo $a ??= $b; // Affiche 10
    echo $a?->nom;  // Affiche la valeur de la propriété nom si $a n'est
pas null
?>
```

Tableau :

- `[]` : utilisé pour définir un tableau ou accéder à un élément du tableau

- `=>` : utilisé pour définir une clé et une valeur dans un tableau associatif

```php
<?php
    $a = [1, 2, 3]; // Définit un tableau
    $a[0];          // Accède à l'élément d'index 0
    $a[1];          // Accède à l'élément d'index 1
    $a[2];          // Accède à l'élément d'index 2
    $a = ["nom" => "John", "age" => 25]; // Définit un tableau associatif
    $a["nom"];      // Accède à l'élément de clé nom
    $a["age"];      // Accède à l'élément de clé age
?>
```

Type :

- `instanceof` : permet de déterminer si un objet est une instance d'une classe particulière ou d'une classe héritée d'une classe spécifique. Il peut également vérifier si un objet implémente une interface particulière.

```php
<?php
    $a = new Personne();
    $a instanceof Personne; // Retourne true si $a est une instance de la classe Personne
?>
```

Exponentiation :

- `**` : élévation à la puissance

```php
<?php
    $a = 2;
    $b = 3;
    echo $a ** $b; // Affiche 8
?>
```

Opérateur de fusion null :

- `??` : retourne la première valeur non null

```php
<?php
    $a = null;
    $b = 5;
    echo $a ?? $b; // Affiche 5
?>
```

Opérateur de coalescence null :

- `??=` : affecte la valeur de droite si la valeur de gauche est null

```php
<?php
    $a = null;
    $b = 5;
    echo $a ??= $b; // Affiche 5
?>
```

Opérateur de navigation null :

- `?->` : accède à une propriété ou une méthode d'un objet, mais ne déclenche pas d'erreur si l'objet est null

```php
<?php
    $a = null;
    $a?->nom; // Affiche null
?>
```

3.5. Structures de contrôle (if, for, while, switch)

Les structures de contrôle permettent de contrôler le flux d'exécution du code. Elles permettent d'exécuter des instructions en fonction de certaines conditions.

1. `if` **/** `else` **/** `elseif` : Permet de tester une condition et d'exécuter du code en fonction du résultat de cette condition.

if :

```php
<?php
    if (condition) {
        // Code à exécuter si la condition est vraie
    }
?>

Exemple concrêt :
<?php
    $age = 25;
    if ($age >= 18) {
        echo "Vous êtes majeur";
    }
?>
```

if / else :

```php
<?php
    if (condition) {
```

```
        // Code à exécuter si la condition est vraie
    } else {
        // Code à exécuter si la condition est fausse
    }
?>
```

Exemple :

```
<?php
    $age = 25;
    if ($age >= 18) {
        echo "Vous êtes majeur";
    } else {
        echo "Vous êtes mineur";
    }
?>
```

if / elseif / else :

```
<?php
    if (condition) {
        // Code à exécuter si la condition est vraie
    } elseif (autre_condition) {
        // Code à exécuter si autre_condition est vraie
    } else {
        // Code à exécuter si aucune des conditions ci-dessus n'est vraie
    }
?>
```

Exemple :

```
<?php
    $age = 25;
    if ($age >= 18) {
        echo "Vous êtes majeur";
    } elseif ($age >= 16) {
        echo "Vous êtes mineur";
    } else {
        echo "Vous êtes trop jeune";
    }
?>
```

2. `for` : Permet d'exécuter une série d'instructions un nombre spécifié de fois.

```php
<?php
    for (initialisation; condition; incrémentation) {
        // Code à exécuter
    }
?>
```

Exemple :

```php
<?php
    for ($i = 0; $i < 10; $i++) {
        echo $i;
    }
?>
```

3. `while` : Permet d'exécuter une série d'instructions tant qu'une condition est vraie.

```php
<?php
    while (condition) {
        // Code à exécuter
    }
?>
```

Exemple :

```php
<?php
    $i = 0;
    while ($i < 10) {
        echo $i;
        $i++; // Affiche 0123456789
    }
?>
```

4. `do while` : Permet d'exécuter une série d'instructions au moins une fois, puis tant qu'une condition est vraie.

```php
<?php
    do {
        // Code à exécuter
    } while (condition);
?>
```

Exemple :

```php
<?php
    $i = 0;  // 1. Initialisation d'une variable $i à 0.

    // 2. Début d'une boucle do-while.
    do {
        echo $i;  // 3. Affiche la valeur courante de $i.

        $i++;     // 4. Incrémente la valeur de $i de 1.

    // 5. Condition de la boucle : tant que $i est inférieur à 10,
continuez à exécuter le corps de la boucle.
    } while ($i < 10);
?>
```

- **Initialisation** : La variable `$i` est initialisée avec la valeur `0`.
- **Début de la boucle `do-while`** : La boucle `do-while` est un type particulier de boucle qui garantit que son corps est exécuté au moins une fois avant de vérifier sa condition.
- Affichage La valeur courante de `$i` est affichée. Lors du premier passage, elle est `0`.
- **Incrémentation** : La valeur de `$i` est augmentée de `1` à chaque itération de la boucle.
- **Condition de boucle** : Après avoir exécuté le corps de la boucle, la condition `$i < 10` est vérifiée.
    - Si la condition est vraie, le corps de la boucle est à nouveau exécuté.
    - Si elle est fausse, la boucle s'arrête.
- Résultat : Le script affiche les nombres de `0` à `9` à la suite, produisant `0123456789`.

5. `foreach` : Permet d'itérer sur les éléments d'un tableau.

```php
<?php
    foreach ($tableau as $valeur) {
        // Code à exécuter
    }
?>
```

Exemple :

```php
<?php
    $hobbies = ["Football", "Cinéma", "Musique"]; // Définit un tableau
    foreach ($hobbies as $hobby) {               // Parcours le tableau
$hobbies
        echo $hobby;                         // Affiche les éléments du
tableau
    }
?>
```

Dans cet exemple, la variable `$hobby` contient la valeur de l'élément courant du tableau à chaque itération de la boucle.

6. `switch` : Une structure de contrôle utilisée pour effectuer différentes actions en fonction de différentes conditions. C'est une alternative à une série de déclarations `if-else`.

```
<?php
    switch (variable) {
        case valeur1:
            // Code à exécuter si la variable vaut valeur1
            break;
        case valeur2:
            // Code à exécuter si la variable vaut valeur2
            break;
        default:
            // Code à exécuter si aucune des valeurs ci-dessus n'est vraie
    }
?>
```

Exemple :

```
<?php
    $age = 25;
    switch ($age) {
        case 18:
            echo "Vous avez 18 ans";
            break;
        case 25:
            echo "Vous avez 25 ans";
            break;
        default:
            echo "Vous n'avez ni 18 ni 25 ans";
    }
?>
```

Dans cet exemple, le script vérifie la valeur de la variable `$age` et exécute le code correspondant à la valeur trouvée. Si aucune des valeurs ne correspond, le code par défaut est exécuté.

PHP Procédural

Le PHP procédural est une approche de programmation qui utilise des fonctions et des instructions pour effectuer des tâches. Il est linéaire et n'utilise pas de concepts avancés tels que des classes ou des objets.

4.1. Concepts de base

Les concepts de base du PHP procédural incluent la définition de variables, l'utilisation des structures de contrôle et l'écriture de fonctions simples.

Exemple :

```php
<?php
  $a = 5;
  $b = 10;
  if ($a < $b) {
    echo "a est inférieur à b";
  }
?>
```

4.2. Fonctions

Les fonctions sont des blocs de code réutilisables conçus pour effectuer une action spécifique.

Exemple :

```php
<?php
  function addition($a, $b) {
    return $a + $b;
  }
  echo addition(5, 10); // Affiche 15
?>
```

Dans ce exemple, on définit une fonction `addition` qui prend deux paramètres `$a` et `$b` et qui retourne la somme de ces deux paramètres. On appelle ensuite cette fonction en lui passant les valeurs `5` et `10` en paramètres.

4.3. Gestion des erreurs

La gestion des erreurs permet d'identifier et de réagir à des conditions d'erreur lors de l'exécution du script.

Exemple :

```php
<?php
  set_error_handler(function($severity, $message, $file, $line) {
    throw new ErrorException($message, 0, $severity, $file, $line);
  });

  try {
    echo 10 / 0;
  } catch (ErrorException $e) {
    echo "Erreur capturée : " . $e->getMessage();
  }
?>
```

Dans ce exemple, on utilise la fonction `set_error_handler` pour définir une fonction de gestion des erreurs personnalisée. Cette fonction est appelée chaque fois qu'une erreur est générée. On utilise ensuite la fonction `try catch` pour capturer l'erreur et afficher un message personnalisé.

4.4. Inclusion de fichiers

L'inclusion de fichiers permet de réutiliser le code PHP stocké dans d'autres fichiers.

Les différentes méthodes d'inclusion de fichiers sont :

- `include` : inclut et exécute le fichier spécifié
- `require` : inclut et exécute le fichier spécifié
- `require_once` : inclut et exécute le fichier spécifié une seule fois

Les directives `include`, `require` `require_once` sont tous des instructions qui permettent d'inclure le contenu d'un fichier dans un autre fichier PHP. La principale différence entre ces instructions réside dans leur comportement lorsqu'un fichier a déjà été inclus.

1. La directive **`require_once`** est utilisée pour inclure et exécuter le contenu d'un fichier. Si le fichier a déjà été inclus dans le script en cours d'exécution, il ne sera pas inclus à nouveau. Cela peut être utile pour éviter de définir plusieurs fois les mêmes variables ou les mêmes fonctions, ce qui pourrait entraîner des erreurs dans le script.

2. La directive **`require`** est similaire à require_once, mais elle n'empêche pas le fichier d'être inclus plusieurs fois. Si le fichier est inclus plusieurs fois, cela peut entraîner des erreurs dans le script, car les variables et les fonctions définies dans ce fichier seront redéfinies plusieurs fois.

3. La directive **`include`** est également similaire à require et `require_once`, mais elle génère une erreur de type "`warning`" si le fichier n'est pas trouvé, au lieu de générer une erreur "fatal" comme c'est le cas avec les deux autres directives. Cela signifie que le script continuera à s'exécuter même si le fichier inclus est introuvable.

En résumé, la directive `require_once` est recommandée lorsque vous souhaitez inclure un fichier qui contient des déclarations importantes pour le script en cours d'exécution, tandis que la directive `include` est plus adaptée lorsque vous souhaitez `inclure` un `fichier` qui n'est pas crucial pour le bon fonctionnement du script.

L'avantage de l'inclusion de fichiers est qu'elle permet de réutiliser le code PHP stocké dans d'autres fichiers. Cela permet de réduire la quantité de code à écrire et de faciliter la maintenance du code.

Exemple avec `include` :

```php
<?php
  include "fichier.php";
```

Exemple avec `require` :

```php
<?php
  require "fichier.php";
```

Exemple avec `require_once` :

```php
<?php
  require_once "data_base.php"; // Inclut le fichier data_base.php
(importnt pour le script)
```

Programmation Orientée Objet (POO) en PHP

La programmation orientée objet est un paradigme de programmation qui utilise des "objets" pour modéliser des données et des comportements. PHP supporte la POO, ce qui permet de créer un code plus modulaire et réutilisable.

5.1. Introduction à la POO

La POO en PHP permet de structurer le code de manière plus naturelle et intuitive, en représentant des concepts et des entités du monde réel.

5.2. Classes et objets

Les classes sont des plans pour créer des objets, qui sont des instances de ces classes.

Exemple :

```php
<?php
class Voiture {
  public $couleur;

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

$maVoiture = new Voiture("rouge");
echo $maVoiture->couleur; // Affiche "rouge"
?>
```

Dans cet exemple, on définit une classe `Voiture` qui contient une propriété `couleur` et une méthode `__construct` qui est appelée lorsqu'un objet est créé à partir de cette classe. On crée ensuite un objet `$maVoiture` à partir de cette classe et on affiche la valeur de la propriété `couleur` de cet objet.

5.3 Les visibilités

En programmation orientée objet en PHP, la visibilité des propriétés et méthodes d'une classe est un concept fondamental qui détermine comment et où les éléments d'une classe peuvent être utilisés. Il existe trois niveaux de visibilité :

- `public` :
  - Les membres déclarés comme `public` sont accessibles de partout.
  - C'est le niveau de visibilité le moins restrictif.

```
class Vehicule {
   public $couleur;

   public function afficherCouleur() {
       echo $this->couleur;
   }
}

$voiture = new Vehicule();
$voiture->couleur = 'rouge';
$voiture->afficherCouleur(); // Affiche 'rouge'
```

- `protected` :
  - Les membres déclarés comme `protected` ne sont accessibles que depuis la classe elle-même et ses classes dérivées.
  - Cela permet de préserver l'encapsulation tout en autorisant l'héritage.

```
class Vehicule {
    protected $vitesseMax;

    protected function obtenirVitesseMax() {
        return $this->vitesseMax;
    }
}

class Voiture extends Vehicule {
    public function afficherVitesseMax() {
        echo $this->obtenirVitesseMax();
    }
}

$voiture = new Voiture();
// $voiture->vitesseMax = 200; // Erreur : Accès protégé
$voiture->afficherVitesseMax(); // Fonctionne si vitesseMax est défini dans
Vehicule
```

- `private` :
  - Les membres déclarés comme `private` sont strictement confinés à la classe qui les a définis.
  - Cela offre un haut niveau d'encapsulation, protégeant les données et les comportements internes de la classe contre les interférences extérieures.

```php
class Vehicule {
    private $numeroSerie;

    private function obtenirNumeroSerie() {
        return $this->numeroSerie;
    }

    public function verifierNumeroSerie($numero) {
        return $numero === $this->obtenirNumeroSerie();
    }
}

$voiture = new Vehicule();
// $voiture->numeroSerie = '123ABC'; // Erreur : Accès privé
$voiture->verifierNumeroSerie('123ABC'); // Fonctionne seulement si
numéroSerie est défini dans le constructeur ou une autre méthode publique
```

Ces exemples illustrent la manière dont les différentes visibilités affectent l'accès aux propriétés et méthodes d'un objet. Ils montrent que `public` permet l'accès de partout, `protected` restreint l'accès aux classes héritières, et `private` restreint l'accès à la classe elle-même.

5.4. Propriétés et méthodes

Les `propriétés` ou `attributs` sont des *variables* dans une classe. Les méthodes sont des fonctions qui définissent des comportements de la classe.

Exemple :

```php
<?php
class Voiture {
  public $couleur;

  public function peindre($couleur) {
    $this->couleur = $couleur;
  }
}

$maVoiture = new Voiture();
$maVoiture->peindre("bleu");
echo $maVoiture->couleur; // Affiche "bleu"
?>
```

Expliquons ce code :

- **`public`** : indique que la propriété ou la méthode est accessible en dehors de la classe.
- **`$couleur`** : est une propriété de la classe `Voiture`.
- **`peindre`** : est une méthode de la classe `Voiture`.
- **`$this`** : est une variable spéciale qui fait référence à l'objet courant.

- `$maVoiture` : est un objet de la classe `Voiture`.
- `->` : est un opérateur qui permet d'accéder aux propriétés et aux méthodes d'un objet.
- `bleu` : est la valeur passée en paramètre à la méthode `peindre`.
- `echo $maVoiture->couleur` : affiche la valeur de la propriété `couleur` de l'objet `$maVoiture`.

Dans cet exemple, on définit une classe `Voiture` qui contient une propriété `couleur` et une méthode `peindre` qui permet de modifier la valeur de cette propriété. On crée ensuite un objet `$maVoiture` à partir de cette classe et on appelle la méthode `peindre` pour modifier la couleur de l'objet.

5.5. Héritage et polymorphisme

- **Héritage** : permet à une classe d'hériter des propriétés et méthodes d'une autre classe. Le polymorphisme est la capacité de traiter des objets de classes différentes de manière interchangeable.

Exemple :

```php
<?php
class Véhicule {
  public function démarrer() {
    echo "Le véhicule démarre";
  }
}

class Voiture extends Véhicule {
  public function démarrer() {
    echo "La voiture démarre";
  }
}

$maVoiture = new Voiture();
$maVoiture->démarrer(); // Affiche "La voiture démarre"
?>
```

Dans cet exemple, on définit une classe `Véhicule` qui contient une méthode `démarrer`. On définit ensuite une classe `Voiture` qui hérite de la classe `Véhicule` et qui redéfinit la méthode `démarrer`. On crée ensuite un objet `$maVoiture` à partir de la classe `Voiture` et on appelle la méthode `démarrer` pour démarrer la voiture.

- **Polyformisme** :

Le polymorphisme est un concept clé de la programmation orientée objet (POO) qui se rapporte à la capacité de différentes classes d'être utilisées de manière interchangeable, même si chacune d'elles pourrait implémenter la même méthode ou propriété de différentes manières. Cela est souvent réalisé via l'héritage ou l'implémentation d'interfaces.

Exemple :

```php
<?php
class Vehicle {
  public function start() {
    echo "Le véhicule démarre";
  }
}

class Car extends Vehicle {
  public function start() {
    echo "La voiture démarre";
  }
}

class Motorcycle extends Vehicle {
  public function start() {
    echo "La moto démarre";
  }
}

function startVehicle(Vehicle $vehicle) {
  $vehicle->start();
}

startVehicle(new Car()); // Affiche "La voiture démarre"
startVehicle(new Motorcycle());    // Affiche "La moto démarre"
?>
```

Dans cet exemple, on définit une classe `Véhicule` qui contient une méthode `démarrer`. On définit ensuite une classe `Voiture` qui hérite de la classe `Véhicule` et qui redéfinit la méthode `démarrer`. On définit également une classe `Moto` qui hérite de la classe `Véhicule` et qui redéfinit la méthode `démarrer`. On définit ensuite une fonction `startVehicle` qui prend un objet de type `Véhicule` en paramètre et qui appelle la méthode `démarrer` de cet objet. On appelle ensuite cette fonction en lui passant un objet `Voiture` et un objet `Moto` en paramètres.

Dans la fonction startVehicle, on fait une `ID` (injection de dépendance). Cela signifie que la fonction prend un objet de type `Véhicule` en paramètre, mais elle peut également prendre un objet de type `Voiture` ou `Moto` car ces deux classes héritent de la classe `Véhicule`.

Une injection de dépendance est un concept de programmation qui permet de rendre le code plus modulaire et réutilisable. Il permet de réduire la quantité de code à écrire et de faciliter la maintenance du code.

5.6. Encapsulation et visibilité

L'encapsulation est le regroupement des données avec les méthodes qui les manipulent. La visibilité (public, protected, private) définit comment ces éléments sont accessibles depuis l'extérieur de la classe.

Exemple :

```php
<?php
class Voiture {
  private $couleur;

  public function getCouleur() {
    return $this->couleur;
  }

  public function setCouleur($couleur) {
    $this->couleur = $couleur;
  }
}

$maVoiture = new Voiture();
$maVoiture->setCouleur("vert");
echo $maVoiture->getCouleur(); // Affiche "vert"
?>
```

Dans cet exemple, on définit une classe `Voiture` qui contient une propriété `couleur` et deux méthodes `getCouleur` et `setCouleur` qui permettent de modifier la valeur de cette propriété. On crée ensuite un objet `$maVoiture` à partir de cette classe et on appelle la méthode `setCouleur` pour modifier la couleur de l'objet. On appelle ensuite la méthode `getCouleur` pour afficher la couleur de l'objet.

Dans cet exemple, on utilise la visibilité `private` pour la propriété `couleur` afin de la protéger contre les interférences extérieures. On utilise ensuite les méthodes `getCouleur` et `setCouleur` pour accéder à cette propriété depuis l'extérieur de la classe.

5.7. Espaces de noms et autoloading

Les espaces de noms permettent de regrouper des classes logiquement et d'éviter les conflits de noms. L'`autoloading` est une fonctionnalité qui charge automatiquement les fichiers de classe sans avoir besoin de `include` ou `require`.

```php
<?php
// Dans le fichier Voiture.php
namespace Vehicules;

class Voiture {
  // ...
}

// Dans un autre fichier
spl_autoload_register(function ($class) {
    include $class . '.php';
});

use Vehicules\Voiture;

$maVoiture = new Voiture();
?>
```

Dans cet exemple, on définit une classe `Voiture` dans un fichier `Voiture.php` et on l'importe dans un autre fichier en utilisant l'instruction `use`. On utilise ensuite la fonction `spl_autoload_register` pour charger automatiquement le fichier `Voiture.php` lorsque la classe `Voiture` est utilisée.

6. Interfaces

Les `interfaces` en PHP définissent des méthodes qui doivent être implémentées par les classes qui les utilisent. Elles ressemblent à des classes abstraites, mais ne peuvent contenir que des déclarations de méthodes et pas leur implémentation. Toutes les méthodes déclarées dans une interface doivent être publiques.

- Une `classe` ne peut pas hériter de plusieurs classes, mais elle peut implémenter plusieurs interfaces.
- Une `interfance` est un contrat qui définit les méthodes qu'une classe doit implémenter. Cela permet de rendre le code plus modulaire et réutilisable.
- Une `interface` est similaire à une classe abstraite, mais elle ne peut contenir que des déclarations de méthodes et pas leur implémentation.
- Une `interface` ne peut pas être instanciée, mais elle peut être implémentée par une classe.
- Une `classe` peut implémenter plusieurs interfaces, mais elle ne peut hériter que d'une seule classe.
- Une `interface` peut hériter d'une ou plusieurs interfaces.
- Une `interface` peut être utilisée comme type de paramètre ou de retour de fonction.
- Une `interface` peut contenir des constantes, mais pas de propriétés.

Déclaration d'une interface

Une interface est déclarée avec le mot-clé `interface`:

```php
interface Mouvement {
    public function demarrer();
    public function arreter();
}
```

Implémentation d'une interface

Une classe implémente une interface en utilisant le mot-clé `implements` :

```php
class Voiture implements Mouvement {
    public function demarrer() {
        // Code pour démarrer la voiture
        echo "La voiture démarre";
    }

    public function arreter() {
        // Code pour arrêter la voiture
        echo "La voiture s'arrête";
```

```
    }
}
```

Utilisation d'une interface

Une fois `l'interface implémentée`, on peut créer des instances de la classe et utiliser les méthodes définies :

```
$maVoiture = new Voiture();
$maVoiture->demarrer(); // Affiche "La voiture démarre"
$maVoiture->arreter();  // Affiche "La voiture s'arrête"
```

Les interfaces sont particulièrement utiles pour fournir un ensemble de méthodes qui doivent être utilisées par différentes classes, permettant ainsi de garantir une cohérence dans la manière dont ces classes sont utilisées.

7. Traits

Les traits en PHP sont un mécanisme de réutilisation de code dans les langages à simple héritage comme PHP. Un trait est similaire à une classe, mais il est destiné à grouper des fonctionnalités de manière fine et cohérente.

Ils permettent de réutiliser des morceaux de code dans plusieurs classes. Les traits peuvent avoir des méthodes et des attributs, et peuvent être utilisés pour éviter certains problèmes de l'héritage multiple.

7.1. Déclaration d'un trait

Un trait est déclaré avec le mot-clé `trait` :

```
trait Loggable {
    public function log($message) {
        echo "Log: $message";
    }
}
```

7.2. Utilisation d'un trait

Un trait est utilisé avec le mot-clé `use` :

```
class User {
    use Loggable; // Utilise le trait Loggable

    public function create() {
        $this->log("Utilisateur créé.");
    }
}
```

7.3. Résolution des conflits de noms

Si deux traits utilisés dans une classe ont une méthode du même nom, une erreur fatale est produite, sauf si une des méthodes est réimplémentée dans la classe qui utilise les traits.

Pour résoudre ce problème, on peut utiliser l'opérateur `insteadof` pour spécifier quelle méthode doit être utilisée :

```
trait Loggable {
    public function log($message) {
        echo "Log from Loggable trait: $message";
    }
}

trait FileLogger {
    public function log($message) {
        echo "Log to a file: $message";
    }
}

class FileStorage {
    use Loggable, FileLogger {
        FileLogger::log insteadof Loggable;
    }
}
```

Dans cet exemple, on définit deux traits `Loggable` et `FileLogger` qui contiennent une méthode `log`. On définit ensuite une classe `FileStorage` qui utilise ces deux traits. On utilise ensuite l'opérateur `insteadof` pour spécifier que la méthode `log` du trait `FileLogger` doit être utilisée.

7.4. Aliases de méthodes de trait

Si deux traits utilisés dans une classe ont une méthode du même nom, on peut utiliser l'opérateur `as` pour spécifier quelle méthode doit être utilisée :

Avec l'`alias`, il est possible de renommer les méthodes des traits pour éviter des conflits ou pour plus de clarté :

```
class User {
    use Loggable, FileLogger {
        FileLogger::log insteadof Loggable;
        Loggable::log as logToScreen;
    }

    public function delete() {
        $this->logToScreen("Utilisateur supprimé.");
    }
```

```
}
```

Dans cet exemple, on définit deux traits `Loggable` et `FileLogger` qui contiennent une méthode `log`. On définit ensuite une classe `FileStorage` qui utilise ces deux traits. On utilise ensuite l'opérateur `as` pour spécifier que la méthode `log` du trait `FileLogger` doit être utilisée sous le nom `logToFile`.

En utilisant les traits, les développeurs PHP peuvent réutiliser des sets de méthodes dans plusieurs classes indépendamment de la hiérarchie de classe.

Fonctions PHP

Lorsque vous travaillez avec les fonctions en PHP, il est essentiel de comprendre comment les créer, les utiliser, gérer les paramètres et les valeurs de retour.

Les fonctions sont des blocs de code réutilisables qui permettent d'effectuer des tâches spécifiques en PHP. Les fonctions permettent de mieux organiser le code et de le rendre plus lisible et plus facile à tester.

8.1. Création et utilisation

Pour créer une fonction en PHP, utilisez le mot-clé `function` suivi du nom de la fonction et d'une paire de parenthèses. Les instructions à exécuter sont placées entre accolades.

```php
function saluer($nom) {
    echo "Bonjour, " . $nom . "!";
}

// Utilisation de la fonction
saluer("Alice");     // Affiche "Bonjour, Alice!"
saluer("Bob");       // Affiche "Bonjour, Bob!"
saluer("Charlie");   // Affiche "Bonjour, Charlie!"
```

Dans cet exemple, on définit une fonction `saluer` qui prend un paramètre `$nom` et qui affiche un message de salutation. On appelle ensuite cette fonction en lui passant différentes valeurs en paramètre.

8.2. Fonctions natives

`PHP` possède une large bibliothèque de fonctions `natives` qui peuvent être utilisées pour diverses opérations comme le travail sur les chaînes de caractères, les tableaux, les fichiers, etc.

Exemple 1 :

```php
<?php
    $nom = "Alice";
    $nom = strtoupper($nom); // Convertit la chaîne en majuscules
    echo $nom;               // Affiche "ALICE"
?>
```

Dans cet exemple, on utilise la fonction `strtoupper` pour convertir la chaîne `$nom` en majuscules.

Exemple 2 :

```
<?php
    $hobbies = ["Football", "Cinéma", "Musique"]; // Définit un tableau
    $hobbies = implode(", ", $hobbies);           // Convertit le tableau
en chaîne
    echo $hobbies;                                // Affiche "Football,
Cinéma, Musique"
?>
```

Dans cet exemple, on utilise la fonction `implode` pour convertir le tableau `$hobbies` en chaîne.

Exemple 3 :

```
$texte = "Bonjour le monde!";
echo str_replace("le monde", "Alice", $texte);
// Affiche "Bonjour Alice!"
```

Dans cet exemple, on utilise la fonction `str_replace` pour remplacer la chaîne `le monde` par `Alice` dans la chaîne `$texte`.

Pour plus d'informations sur les fonctions natives de PHP, consultez la documentation officielle.

8.3. Passage de paramètres

Les paramètres sont des variables utilisées pour passer des valeurs aux fonctions. En PHP, les paramètres peuvent être passés par valeur (comportement par défaut) ou par référence en utilisant le symbole `&`.

Exemple 1 :

```
function ajouter_un(&$nombre) {
    $nombre++;
}

$compteur = 5;
ajouter_un($compteur);
echo $compteur; // Affiche 6
```

Dans cet exemple, on définit une fonction `ajouter_un` qui prend un paramètre `$nombre` par référence et qui incrémente sa valeur de `1`. On appelle ensuite cette fonction en lui passant la variable `$compteur` en paramètre.

8.4. Retour de valeurs

Une fonction peut retourner une valeur à l'aide de l'instruction `return`. Dès que return est exécuté, la fonction est terminée et la valeur est retournée à l'appelant.

Exemple :

```
function addition($a, $b) {
    return $a + $b; // Retourne la somme de $a et $b
}

echo addition(5, 10); // Affiche 15
```

Dans cet exemple, on définit une fonction `addition` qui prend deux paramètres `$a` et `$b` et qui retourne la somme de ces deux paramètres. On appelle ensuite cette fonction en lui passant les valeurs `5` et `10` en paramètres.

Gestion des formulaires

La gestion des formulaires est une partie essentielle du développement Web avec PHP

Les formulaires sont le moyen standard de recueillir des informations de la part des utilisateurs sur le Web. PHP offre des méthodes efficaces pour récupérer, valider et sécuriser les données de formulaire.

9.1. Récupération des données

Les données de formulaire sont généralement envoyées via des méthodes `GET` ou `POST` et sont accessibles en PHP via les superglobales `$_GET` et `$_POST`.

Les variables superglobales sont des variables prédéfinies qui sont toujours accessibles, quel que soit le contexte. Elles sont accessibles de partout dans le script, y compris à l'intérieur des fonctions.

```
// Récupération des données d'un formulaire avec la méthode GET
$nom = $_GET['nom'];

// Récupération des données d'un formulaire avec la méthode POST
$email = $_POST['email'];
```

9.2. Validation des données

Avant de traiter ou de stocker des données, il est crucial de les valider pour s'assurer qu'elles répondent à certains critères.

Exemple 1 :

```
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "L'adresse email est valide.";
} else {
```

```
    echo "L'adresse email n'est pas valide.";
}
```

Dans cet exemple, on utilise la fonction `filter_var` pour valider l'adresse email `$email`. Si l'adresse email est valide, on affiche un message de confirmation. Sinon, on affiche un message d'erreur.

Exemple 2 :

```
if (filter_var($url, FILTER_VALIDATE_URL)) {
    echo "L'URL est valide.";
} else {
    echo "L'URL n'est pas valide.";
}
```

Dans cet exemple, on utilise la fonction `filter_var` pour valider l'URL `$url`. Si l'URL est valide, on affiche un message de confirmation. Sinon, on affiche un message d'erreur.

Exemple 3 :

```
if (filter_var($ip, FILTER_VALIDATE_IP)) {
    echo "L'adresse IP est valide.";
} else {
    echo "L'adresse IP n'est pas valide.";
}
```

Dans cet exemple, on utilise la fonction `filter_var` pour valider l'adresse IP `$ip`. Si l'adresse IP est valide, on affiche un message de confirmation. Sinon, on affiche un message d'erreur.

Pour plus d'informations sur les filtres, consultez la documentation officielle.

9.3. Sécurisation des entrées utilisateur

Ils ne faut jamais faire confiance aux données envoyées par les utilisateurs. Il est important d'assainir et de sécuriser ces données avant de les utiliser.

Exemple 1 :

```
$nom = htmlspecialchars($_POST['nom']);
```

Dans cet exemple, on utilise la fonction `htmlspecialchars` pour convertir les caractères spéciaux en entités HTML. Cela permet d'éviter les attaques XSS (Cross-Site Scripting).

Exemple 2 :

```
$nom = filter_var($_POST['nom'], FILTER_SANITIZE_STRING);
```

Dans cet exemple, on utilise la fonction `filter_var` pour supprimer les balises HTML et les caractères spéciaux de la chaîne `$nom`. Cela permet d'éviter les attaques XSS (Cross-Site Scripting).

Lorsque l'utilisateur insère par exemple du code HTML dans un champ de formulaire, il est important de supprimer les balises HTML et les caractères spéciaux pour éviter les attaques XSS (Cross-Site Scripting). C'est en ce moment l'utilisation de la fonction `filter_var` est très importante.

`filter_var` prend en paramètre la variable à filtrer et le filtre à appliquer qui est ici `FILTER_SANITIZE_STRING`.

`FILTER_SANITIZE_STRING` qui permet de supprimer les balises HTML et les caractères spéciaux de la chaîne `$nom`.

Exemple 3 : Assainissement d'une chaîne pour l'échapper avant de l'insérer dans une base de données

```
// Assainissement d'une chaîne pour l'échapper avant de l'insérer dans une
base de données
$nom_securise = mysqli_real_escape_string($connection, $nom);

// Utilisation de htmlspecialchars pour éviter les injections XSS
$commentaire_securise = htmlspecialchars($commentaire);
```

La gestion sûre des formulaires est essentielle pour protéger votre application contre les attaques courantes comme l'injection SQL et le cross-site scripting (XSS).

PHP et bases de données

La manipulation des bases de données est une opération courante dans le développement web. PHP fournit des moyens pour se connecter à différentes bases de données, effectuer des requêtes et gérer les données de manière sécurisée.

10.1. Connexion à une base de données

Pour interagir avec une base de données, PHP doit d'abord établir une connexion en utilisant des extensions comme `mysqli` ou `PDO`. Avec PHP, il existe deux méthodes pour se connecter à une base de données :

- **`mysqli`** : est une extension orientée objet qui permet de se connecter à une base de données MySQL.
- **`PDO`** : est une extension orientée objet qui permet de se connecter à plusieurs types de bases de données.

Connexion à une base de données MySQL avec mysqli

```
// Connexion à une base de données MySQL avec mysqli
```

```php
$serveur = "localhost";
$utilisateur = "nom_utilisateur";
$mot_de_passe = "mot_de_passe";
$nom_base_de_donnees = "nom_base_de_donnees";

$connection = new mysqli($serveur, $utilisateur, $mot_de_passe,
$nom_base_de_donnees); // Crée un objet de connexion à la base de données

if ($connection->connect_error) {
    die("Erreur de connexion: " . $connection->connect_error); // Affiche
l'erreur de connexion
}
echo "Connexion réussie";
```

Expliquons ce code en détail :

- `$serveur` : contient le nom du serveur de base de données dont `localhost`.
- `$utilisateur` : contient le nom d'utilisateur de la base de données. Par defaut c'est `root`.
- `$mot_de_passe` : contient le mot de passe de la base de données. Par defaut, l'utilisateur c'est `root` sans mot de passe.
- `$nom_base_de_donnees` : contient le nom de la base de données.
- `$connection` : contient l'objet de connexion à la base de données. C'est le `DNS` (Data Source Name) de la base de données.
- `$connection->connect_error` : contient le message d'erreur de connexion à la base de données.
- `echo "Connexion réussie";` : On affiche le message de connexion réussie à la base de données.

Connexion à une base de données MySQL avec PDO

```php
// Connexion à une base de données MySQL avec PDO
$serveur = "localhost";
$utilisateur = "nom_utilisateur";
$mot_de_passe = "mot_de_passe";
$nom_base_de_donnees = "nom_base_de_donnees";

try {
    $connection = new
PDO("mysql:host=$serveur;dbname=$nom_base_de_donnees", $utilisateur,
$mot_de_passe); // Crée un objet de connexion à la base de données
    $connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Configure le mode d'erreur de PDO sur Exception
    echo "Connexion réussie";
} catch(PDOException $e) {
    die("Erreur de connexion: " . $e->getMessage()); // Affiche l'erreur de
connexion
}
```

Dans ce code, on utilise le bloc `try...catch` pour gérer les erreurs de connexion à la base de données. On utilise ensuite la méthode `setAttribute` pour configurer le mode d'erreur de PDO sur Exception.

10.2. Requêtes SQL avec PHP

Après avoir établi une connexion, vous pouvez exécuter des requêtes SQL pour interagir avec la base de données.

Exécution d'une requête SQL avec mysqli

On suppose qu'on a défini une connexion à la base de données dans un fichier `connexion.php` comme ceci :

```
// Exécution d'une requête SQL avec mysqli

require_once 'connexion.php'; // Inclut le fichier de connexion à la base de données

$sql = "SELECT * FROM utilisateurs";

$resultat = $connection->query($sql); // Exécute la requête SQL et retourne un résultat

if ($resultat->num_rows > 0) { // Vérifie si le résultat contient au moins une ligne
    while($ligne = $resultat->fetch_assoc()) { // Parcours chaque ligne du résultat
        echo "Nom: " . $ligne["nom"] . " - Email: " . $ligne["email"] . "<br>";
    }
} else {
    echo "Aucun résultat";
}
```

Expliquons ce code en détail :

- `require_once 'connexion.php';` : inclut le fichier de connexion à la base de données.
- `$sql` : contient la requête SQL à exécuter.
- `$resultat` : contient le résultat de la requête SQL.
- `$resultat->num_rows` : contient le nombre de lignes du résultat.
- `$resultat->fetch_assoc()` : retourne la ligne suivante du résultat sous forme de tableau associatif.
- `echo "Nom: " . $ligne["nom"] . " - Email: " . $ligne["email"] . "<br>";` : affiche le nom et l'email de chaque utilisateur.
- `echo "Aucun résultat";` : affiche un message si le résultat est vide.

Exécution d'une requête SQL avec PDO

On suppose qu'on a défini une connexion à la base de données dans un fichier `connexion.php` comme ceci :

```php
// Exécution d'une requête SQL avec PDO

require_once 'connexion.php'; // Inclut le fichier de connexion à la base de données

$sql = "SELECT * FROM utilisateurs";

$resultat = $connection->query($sql); // Exécute la requête SQL et retourne un résultat

if ($resultat->rowCount() > 0) { // Vérifie si le résultat contient au moins une ligne
    while($ligne = $resultat->fetch()) { // Parcours chaque ligne du résultat
        echo "Nom: " . $ligne["nom"] . " - Email: " . $ligne["email"] . "<br>";
    }
} else {
    echo "Aucun résultat";
}
```

10.3. Sécurité des bases de données

La sécurité est primordiale lors de l'interaction avec les bases de données pour éviter les injections SQL et protéger les données.

Injection SQL

L'injection SQL est une technique d'attaque qui consiste à insérer du code SQL malveillant dans une requête SQL via les entrées utilisateur.

Exemple :

```php
// Injection SQL
$sql = "SELECT * FROM utilisateurs WHERE email = '" . $_POST['email'] . "' AND mot_de_passe = '" . $_POST['mot_de_passe'] . "'";
```

Dans cet exemple, on utilise les entrées utilisateur `$_POST['email']` et `$_POST['mot_de_passe']` dans une requête SQL sans les assainir. Cela permet à un attaquant d'injecter du code SQL malveillant dans la requête.

Pour éviter les injections SQL, il est important d'assainir les entrées utilisateur avant de les utiliser dans une requête SQL avec des requêtes préparées.

Requêtes préparées

Une requête préparée est une requête SQL qui est préparée et compilée par le serveur de base de données avant d'être exécutée. Elle permet d'éviter les injections SQL et d'améliorer les performances en exécutant

plusieurs fois la même requête.

Exemple :

```
// Requête préparée
$sql = "SELECT * FROM utilisateurs WHERE email = ? AND mot_de_passe = ?";
$requete = $connection->prepare($sql); // Prépare la requête SQL
$requete->bind_param("ss", $_POST['email'], $_POST['mot_de_passe']); // Lie
les paramètres à la requête
$requete->execute(); // Exécute la requête
$resultat = $requete->get_result(); // Récupère le résultat de la requête
```

- `$sql` : contient la requête SQL à exécuter.
- `$requete` : contient la requête préparée.
- `$requete->bind_param("ss", $_POST['email'], $_POST['mot_de_passe']);` : lie les paramètres à la requête.
- `$requete->execute();` : exécute la requête.
- `$resultat = $requete->get_result();` : récupère le résultat de la requête.

Les deux `ss` dans la méthode `bind_param` indiquent que les deux paramètres sont des chaînes de caractères. Si les paramètres étaient des entiers, on aurait utilisé deux `ii` à la place.

ou

```
// Utilisation des requêtes préparées avec PDO pour prévenir les injections
SQL

require_once 'connexion.php'; // Inclut le fichier de connexion à la base
de données

$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$stmt = $pdo->prepare("INSERT INTO utilisateurs (nom, email) VALUES (:nom,
:email)");
$stmt->bindParam(':nom', $nom);
$stmt->bindParam(':email', $email);

$nom = "John Doe";
$email = "john.doe@example.com";
$stmt->execute();
```

Expliquons ce code en détail :

- `$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);` : configure le mode d'erreur de PDO sur Exception.
- `$stmt = $pdo->prepare("INSERT INTO utilisateurs (nom, email) VALUES (:nom, :email)");` : prépare la requête SQL.
- `$stmt->bindParam(':nom', $nom);` : lie le paramètre `:nom` à la variable `$nom`.

- `$stmt->bindParam(':email', $email);` : lie le paramètre `:email` à la variable `$email`.
- `$nom = "John Doe";` : définit la valeur de la variable `$nom`.
- `$email = "john.doe@example.com";` : définit la valeur de la variable `$email`.
- `$stmt->execute();` : exécute la requête.

Il est essentiel de toujours utiliser des requêtes préparées ou des mécanismes similaires lors de l'insertion ou de la mise à jour de données pour prévenir les vulnérabilités de sécurité.

Sessions et cookies

Les sessions et les cookies sont deux mécanismes fondamentaux pour maintenir des données entre différentes requêtes et visites des utilisateurs sur un site web. PHP fournit des fonctionnalités intégrées pour gérer ces données de manière efficace et sécurisée.

11.1. Gestion des sessions

Les sessions en PHP permettent de stocker des informations sur le serveur pour les utiliser tout au long de la session utilisateur.

```
// Démarrage d'une nouvelle session ou reprise d'une session existante
session_start();

// Stockage d'informations en session
$_SESSION['utilisateur'] = 'nom_utilisateur';

// Accès à l'information stockée
echo 'Bienvenue, ' . $_SESSION['utilisateur'];

// Destruction de la session
session_destroy();
```

Dans cet exemple, on démarre une nouvelle session ou on reprend une session existante. On stocke ensuite le nom de l'utilisateur dans la variable `$_SESSION['utilisateur']`. On affiche ensuite un message de bienvenue à l'utilisateur. On détruit ensuite la session.

11.2. Utilisation des cookies

Les cookies sont des données stockées côté client et envoyées au serveur avec chaque requête HTTP. Ils sont utiles pour stocker les préférences utilisateur, les informations d'identification, etc.

```
// Création d'un cookie
setcookie("utilisateur", "nom_utilisateur", time() + 3600); // Expire après
1 heure

// Accès à un cookie
if(isset($_COOKIE['utilisateur'])) {
    echo 'Utilisateur: ' . $_COOKIE['utilisateur'];
}
```

```
// Suppression d'un cookie
setcookie("utilisateur", "", time() - 3600); // Expire dans 1 heure
```

Il est important de gérer les sessions et les cookies avec soin pour protéger la vie privée des utilisateurs et la sécurité des données. Toujours valider et nettoyer les données de session et de cookies pour éviter les attaques de type cross-site scripting (XSS) ou autres vulnérabilités de sécurité.

Sécurité en PHP

La sécurité en PHP est un aspect essentiel à prendre en compte lors de la création d'applications Web. La prévention des injections SQL, des attaques XSS et la gestion sécurisée des mots de passe sont des thèmes critiques.

La sécurité est primordiale dans le développement web. PHP offre plusieurs mécanismes pour aider les développeurs à sécuriser leurs applications contre les attaques les plus courantes.

12.1. Prévention des injections SQL

Les injections SQL peuvent compromettre la sécurité de votre base de données. Utiliser les requêtes préparées avec des statements PDO ou MySQLi est une des meilleures pratiques pour prévenir ce type d'attaque.

```
// Utilisation de PDO pour prévenir les injections SQL
$pdo = new PDO('mysql:host=exemple_host;dbname=exemple_db', 'utilisateur',
'motdepasse');
$statement = $pdo->prepare('SELECT * FROM utilisateurs WHERE nom = :nom');
$statement->execute(['nom' => $nomUtilisateur]);
$resultat = $statement->fetchAll();
```

Dans cet exemple, on utilise PDO pour prévenir les injections SQL. On utilise la méthode `prepare` pour préparer la requête SQL. On utilise ensuite la méthode `execute` pour exécuter la requête SQL en passant le nom d'utilisateur en paramètre. On utilise ensuite la méthode `fetchAll` pour récupérer le résultat de la requête.

12.2. Prévention des attaques XSS

Les attaques par scripts intersites (XSS) se produisent lorsque des scripts malveillants sont injectés dans des pages web. Utiliser la fonction htmlspecialchars() pour échapper à l'entrée des utilisateurs est une méthode pour prévenir les XSS.

```
// Échappement de la sortie pour prévenir les XSS
echo htmlspecialchars($string, ENT_QUOTES, 'UTF-8');
```

12.3. Gestion sécurisée des mots de passe

PHP fournit des fonctions pour créer et vérifier les mots de passe de manière sécurisée. Utilisez `password_hash()` pour créer un hash de mot de passe et `password_verify()` pour le vérifier. Pour des raisons de sécurité, il est important de ne jamais stocker les mots de passe en clair dans la base de données.

```php
// Création d'un hash sécurisé pour un mot de passe
$motDePasseHash = password_hash('motDePasseSecret', PASSWORD_DEFAULT);

// Vérification du mot de passe saisi contre le hash
if (password_verify('motDePasseSecret', $motDePasseHash)) {
    // Le mot de passe est correct
} else {
    // Le mot de passe est incorrect
}
```

Il existe différentes méthodes de hashage de mots de passe :

- `PASSWORD_DEFAULT` : utilise l'algorithme `bcrypt` (par défaut depuis PHP 5.5.0).
- `PASSWORD_BCRYPT` : utilise l'algorithme `bcrypt`.
- `PASSWORD_ARGON2I` : utilise l'algorithme `Argon2i` (disponible depuis PHP 7.2.0).
- `PASSWORD_ARGON2ID` : utilise l'algorithme `Argon2id` (disponible depuis PHP 7.3.0).
- `PASSWORD_BCRYPT` : utilise l'algorithme `bcrypt`.
- `md5` : utilise l'algorithme `md5' (*déconseillé*).

Toujours rester informé des meilleures pratiques de sécurité et les mettre en œuvre pour protéger vos applications PHP des menaces potentielles.

Frameworks PHP populaires

- **Laravel** : est un framework PHP open-source basé sur le modèle MVC (Modèle-Vue-Contrôleur) pour le développement d'applications web. Site officiel : https://laravel.com/
- **Symfony** : est un framework PHP open-source basé sur le modèle MVC (Modèle-Vue-Contrôleur) pour le développement d'applications web. Site officiel : https://symfony.com/
- **CodeIgniter** : est un framework PHP open-source basé sur le modèle MVC (Modèle-Vue-Contrôleur) pour le développement d'applications web. Site officiel : https://codeigniter.com/

Et bien d'autres encore...

Ces frameworks PHP sont très populaires et sont utilisés par de nombreuses entreprises pour développer des applications web.

Débogage et optimisation

Pour garantir la robustesse et l'efficacité des applications PHP, le débogage et l'optimisation du code sont des étapes incontournables du processus de développement.

14.1. Outils de débogage

La phase de débogage et d'optimisation est cruciale dans le cycle de développement d'une application pour assurer sa qualité, sa performance et sa maintenabilité.

Des outils comme `Xdebug` et `Zend Debugger` permettent aux développeurs de parcourir leur code, de mettre des points d'arrêt et d'inspecter les variables pour identifier les bugs et les problèmes de logique.

```
// Exemple de code avec un point d'arrêt Xdebug
if ($user->isAuthorized()) {
    xdebug_break(); // Un point d'arrêt est défini ici
    // Code additionnel pour la démonstration
}
```

14.2. Optimisation des performances

L'optimisation peut impliquer la mise en cache de données fréquemment accédées, l'utilisation de compilateurs JIT comme OPCache, et la révision de l'architecture du code pour améliorer l'efficience.

```
// Exemple d'activation de OPCache dans php.ini
opcache.enable=1 // Active OPCache
opcache.memory_consumption=128 // Taille du cache en Mo
opcache.interned_strings_buffer=8 // Taille du cache pour les chaînes de
caractères
opcache.max_accelerated_files=4000 // Nombre maximum de fichiers pouvant
être mis en cache
opcache.revalidate_freq=60 // Durée de vie du cache en secondes
opcache.fast_shutdown=1 // Pour une meilleure performance
```

Une approche proactive du débogage et de l'optimisation permet non seulement de réduire les temps d'arrêt et d'améliorer l'expérience utilisateur, mais aussi de réduire les coûts de serveur et de maintenir le code plus facilement sur le long terme.

Conclusion

En parcourant l'histoire, les caractéristiques et les divers aspects de PHP, nous avons pu constater que, malgré ses détracteurs et les défis auxquels il fait face, PHP reste une pierre angulaire du développement web.

PHP a évolué bien au-delà de ses racines modestes pour devenir un langage puissant, capable de propulser des applications web de toutes tailles. Avec ses frameworks modernes tels que Laravel et Symfony, sa capacité à s'intégrer avec divers systèmes de gestion de base de données, et son écosystème riche en ressources et en communauté, PHP maintient sa position comme une option fiable pour de nombreux développeurs et entreprises.

Les discussions sur la sécurité, l'optimisation, et le débogage montrent que PHP est non seulement adaptable mais, aussi soucieux des meilleures pratiques et des normes modernes de développement. La capacité du langage à innover et à s'adapter aux nouvelles exigences du marché est un témoignage de sa maturité et de sa longévité.

En tant que langage dynamique et en constante évolution, PHP continuera de jouer un rôle essentiel dans l'avenir du développement web. Les développeurs qui maîtrisent PHP et ses nuances se trouveront bien équipés pour créer des solutions web robustes, évolutives et performantes.

Ainsi, PHP reste un choix solide pour le développement web, soutenu par une communauté globale et une histoire riche en innovations. La continuité de son développement laisse présager un avenir brillant pour ce langage versatile.

Références

- PHP
- Wikipédia
- PHP 8.0