Découverte du PHP

Présentation

Qu'est-ce que PHP ?

PHP (Hypertext Preprocessor) est un langage de programmation serveur utilisé pour créer des sites web dynamiques.

Histoire et évolution de PHP

Il a été créé en 1994 par Rasmus Lerdorf et a évolué au fil des ans pour devenir l'un des langages de programmation les plus populaires pour le développement web.

Utilisation de PHP dans le développement web

PHP est utilisé pour créer des sites web dynamiques, des applications web, des API, etc.

Installation et configuration

Installation de PHP 8.x

Pour commencer à utiliser PHP, vous devez l'installer sur votre système d'exploitation. Vous pouvez télécharger la dernière version de PHP sur le site officiel de PHP.

Dans ce cours, nous faisons uniquement du PHP impératif : pas besoin d'installer d'environnement (XAMP, Lighttpd, etc.) dans un premier temps. Nous téléchargeons uniquement l'archive compressée à l'adresse suivante sous Windows https://windows.php.net/download/. Pour Mac il suffit d'utiliser le gestionnaire de paquet Homebrew, et sous Linux (Ubuntu/Debian), apt ou snap (plus flexible, idéal pour tester).

Une fois l'archive décompresser à l'endroit voulu sous Windows, il suffit de mettre à jour la variable d'environnement PATH pour qu'elle pointe sur le dossier contenant l'exécutable php.exe.

```
$ php -v
PHP 8.4.6 (cli) (built: Apr  9 2025 09:45:13) (NTS Visual C++ 2022 x6
4)
Copyright (c) The PHP Group
Zend Engine v4.4.6, Copyright (c) Zend Technologies
```

Cette simple commande permet de contrôler que la commande `php` est bien disponible depuis un terminal, ainsi que la version de PHP.

Configuration de l'environnement de développement

Nous utilisons VSCode dans ce cours de découverte (PHPStorm est le plus utilisé mais il faut une licence).

Pour coder confortablement en PHP on peut installer les extensions suivantes :

- **PHP Intelephense** : Autocomplétion et navigation pour PHP, avec des suggestions de code et de la documentation.
- **PHP Namespace** : Ajoute ou résout les espaces de noms PHP automatiquement dans les fichiers.
- **PHP Xdebug** : Intégration de Xdebug pour déboguer PHP directement dans VS Code.
- **PHP DocBlocker** : Génère et formate les blocs de documentation PHP (docblocks).
- **Auto Close Tag** : Ferme automatiquement les balises HTML/XML lors de la saisie.
- **Auto Rename Tag** : Renomme automatiquement la balise fermante lorsqu'une balise ouvrante est modifiée.
- **Todo Tree** : Affiche les commentaires TODO dans un arbre pour mieux gérer les tâches.
- **Error Lens** : Surligne les erreurs et avertissements directement dans l'éditeur.
- **indent-rainbow** : Colore les niveaux d'indentation pour améliorer la lisibilité du code.
- **Bracket Pair Color DLW** : Colorie les paires de parenthèses pour faciliter la lecture du code.
- **Faker.js** : Génère des données factices pour les tests (noms, adresses, etc.).

Démarrage : de la syntaxe aux conditions

Balises PHP

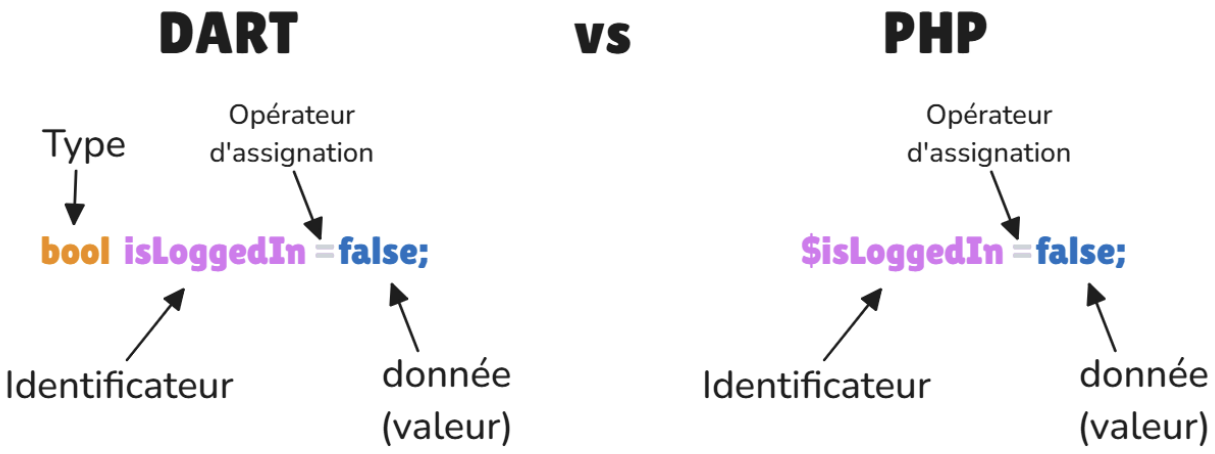
La syntaxe de PHP est délimitée par des balises . Voici un exemple de code PHP :

```php
<?php

  // code PHP ici

?> // Non nécessaire si le fichier contient exclusivement du PHP
```

Variables et types de données



PHP est un langage à typage faible (conversion automatique) et dynamique (déterminé à l'exécution). Depuis PHP 7, on peut activer un typage plus strict `declare(strict_types=1);` .

Les variables en PHP sont utilisées pour stocker des valeurs.

Vous pouvez déclarer une variable en utilisant le signe $ suivi du nom de la variable.
Les types de données en PHP incluent les entiers, les chaînes de caractères, les booléens, les tableaux, etc.

```php
<?php
  $name = 'Coco'; // chaîne de caractères
  $age = 76; // entier
```

```php
    $admin = true; // booléen
?>
```

Opérateurs

En PHP, les **opérateurs** sont utilisés pour effectuer diverses opérations sur des variables et des valeurs. Il en existe plusieurs types, chacun ayant des fonctionnalités spécifiques.

Voici un aperçu des principaux opérateurs en PHP :

1. **Opérateurs arithmétiques**

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

```php
<?php
$a = 10;
$b = 5;

echo $a + $b; // 15
echo $a - $b; // 5
echo $a * $b; // 50
echo $a / $b; // 2
echo $a % $b; // 0
?>
```

2. **Opérateurs de comparaison (permettent de comparer deux valeurs)**

- `==` : égalité (valeurs égales)
- `===` : égalité stricte (valeurs et types égaux)
- `!=` ou `<>` : inégalité
- `!==` : inégalité stricte (valeurs ou types différents)
- `<` : inférieur à
- `>` : supérieur à
- `<=` : inférieur ou égal à
- `>=` : supérieur ou égal à

```php
<?php
$a = 10;
$b = '10';

var_dump($a == $b);  // true (comparaison des valeurs)
var_dump($a === $b); // false (comparaison des valeurs et types)
?>
```

3. **Opérateurs logiques (permettent de combiner des conditions)**

- `&&` : ET logique
- `||` : OU logique
- `!` : NON logique

```php
<?php
$a = true;
$b = false;

var_dump($a && $b); // false
var_dump($a || $b); // true
var_dump(!$a);
?>
```

4. **Opérateurs d'affectation (utilisés pour affecter des valeurs aux variables)**

- `=` : affectation simple
- `+=` : addition et affectation
- `-=` : soustraction et affectation
- `*=` : multiplication et affectation
- `/=` : division et affectation
- `%=` : modulo et affectation

```php
<?php
$a = 10;
$a += 5; // $a = $a + 5
echo $a; // 15
?>
```

5. **Opérateurs de concaténation (utilisés pour assembler des chaînes de caractères)**

- `.` : concaténation (ajoute une chaîne à une autre)
- `.=` : concaténation avec affectation

```php
<?php
$firstName = 'John';
$lastName = 'Doe';

$fullName = $firstName . ' ' . $lastName; // John Doe
echo $fullName;

$fullName .= ' Jr.'; // Ajoute ' Jr.' à $fullName
echo $fullName; // John Doe Jr.
?>
```

6. **Opérateurs d'incrémentation et de décrémentation (augmentent ou diminuent une variable)**

- `++` : incrémentation (ajoute 1)
- `--` : décrémentation (soustrait 1)

```php
<?php
$a = 5;
$a++; // $a = $a + 1
echo $a; // 6

$a--; // $a = $a - 1
echo $a; // 5
?>
```

7. **Opérateurs de type (permettent de vérifier le type d'une variable ou de la forcer à un autre type)**

- `is_*` : fonction qui permet de vérifier si une variable est de certains types (ex. `is_int()`, `is_string()`)
- `(type)` : cast explicite (conversion de type)

```php
<?php
$a = 10;
var_dump(is_int($a));  // true

$b = (float) $a; // Convertit $a en float
```

```php
var_dump(is_float($b));  // true
?>
```

8. **Opérateurs de tableau**

- `[]` : pour accéder ou ajouter une valeur dans un tableau
- `array_key_exists()` : vérifie si une clé existe dans un tableau
- `in_array()` : vérifie si une valeur existe dans un tableau
- `array_merge()` : fusionne des tableaux

```php
<?php
$array = ['a' => 1, 'b' => 2];
echo $array['a']; // 1

if (array_key_exists('a', $array)) {
    echo "La clé 'a' existe.";
}
?>
```

9. **Opérateurs ternaires (forme compacte de `if` en une seule ligne)**

- `condition ? valeur_si_vrai : valeur_si_faux`

```php
<?php
$age = 20;
echo $age >= 18 ? 'Adulte' : 'Mineur'; // Adulte
?>
```

Découverte des structures conditionnelles

Les structures de contrôle en PHP sont utilisées pour gérer le flux d'exécution du code. Dans cette section, nous aborderons les conditions, qui permettent au programme d'ajuster son comportement en fonction de critères spécifiques. Dans le paragraphe suivant, nous explorerons les itérations, qui permettent au programme de répéter certaines actions selon des critères définis.

Il y a 4 types de structures conditionnelles en PHP :

- `if` : Permet d'exécuter un bloc de code si une condition est vraie.
- `else` : Permet d'exécuter un bloc de code si la condition dans le `if` est fausse.
- `elseif` : Permet de vérifier une autre condition si la condition initiale du `if` est fausse.

- `switch` : Permet de tester une variable contre plusieurs valeurs possibles et d'exécuter un bloc de code correspondant à une valeur donnée.
  Chacune de ces structures permet de diriger l'exécution du programme en fonction de certaines conditions.

Voici un exemple d'utilisation du trio `if`, `elseif`, `else` :

```php
<?php
$role = 'ROLE_ADMIN';

if ($role == 'ROLE_ADMIN') {
    echo "Tu peux ajouter/supprimer/éditer/copier/remplacer.";
} elseif ($role == 'ROLE_EDITOR') {
    echo "Tu peux seulement éditer.";
} else {
    echo "Accès interdit.";
}
?>
```

Et voici un exemple d'utilisation de `switch` :

```php
<?php
$friend = 'coco';

switch ($friend) {
    case 'alice':
    case 'bob':
        echo "Ton ami est Alice ou Bob.";
        break;
    case 'coco':
        echo "Ton ami est Coco.";
        break;
    default:
        echo "Ton ami n'est ni Alice ni Coco.";
        break;
}
?>
```

En PHP, on ne peut pas labeliser une structure de contrôle de la même façon qu'en JavaScript mais il est facile de pallier cela :

```php
<?php
$friend = 'bob';
$age = 30;
```

```php
echo "Début du programme.\n";

// Label de départ
start:

switch ($friend) {
    case 'alice':
        echo "Ton ami est Alice.\n";
        break;
    case 'bob':
        echo "Ton ami est Bob.\n";
        goto end; // Saute à la fin du programme si $friend est 'bob'
        break;
    default:
        echo "Ton ami est inconnu.\n";
        break;
}

echo "Cette ligne ne sera pas atteinte si $friend est 'bob'.\n";

// Label de fin
end:
echo "Fin du programme.\n";
?>
```

Aller plus loin avec les structures conditionnelles

1. **Opérateurs conditionnels ternaires**
   Les opérateurs ternaires offrent une manière concise d'écrire des conditions. Ils sont souvent utilisés pour assigner une valeur à une variable en fonction d'une condition. Cela permet de remplacer des blocs `if` / `else` simples par une seule ligne de code.

```php
<?php
`$age = 20; $status = ($age >= 18) ? 'Adulte' : 'Mineur';  // Retourne
'Adulte'`
?>
```

2. **Opérateur Null Coalescent** (`??`) :
   Introduit dans PHP 7, l'opérateur `??` permet de vérifier si une variable est définie et non `null` . Cela simplifie et rend le code plus lisible, notamment quand on travaille avec des entrées de formulaires ou des paramètres.

Remarque : avant cet opérateur, on utilisait couramment un bloc conditionnelle avec `isset()`.

```php
<?php
if (isset($_GET['name'])) {
    $name = $_GET['name'];
} else {
    $name = 'Invité';
}

// VS

$name = $_GET['name'] ?? 'Invité';  // Si 'name' est non défini, 'Invité' est utilisé`
?>
```

3. **Conditions imbriquées** :
   Les conditions peuvent être imbriquées les unes dans les autres. Cela est particulièrement utile pour des scénarios complexes où plusieurs critères doivent être vérifiés.
   Cependant, l'imbrication excessive des conditions peut rendre le code difficile à suivre, il est donc préférable de limiter leur utilisation ou de les restructurer avec des fonctions.

```php
<?php
$age = 25;
$gender = 'femme';

if ($age >= 18) {
    if ($gender == 'femme') {
        echo "Adulte, femme";
    } else {
        echo "Adulte, homme";
    }
} else {
    echo "Mineur";
}
?>
```

4. **Conditionnelle avec `match` (PHP 8.0+)** :
   Le `match` est une alternative moderne au `switch`. Il permet de comparer une expression à plusieurs valeurs possibles, mais avec plus de flexibilité et de sécurité. Il ne permet pas de "tomber à travers" les cases comme avec `switch`.
   Le `match` est strict sur les types, ce qui évite des comportements inattendus que l'on peut parfois rencontrer avec `switch`.

```php
<?php
$color = 'rouge';

echo match($color) {
    'rouge' => 'La couleur est rouge',
    'bleu' => 'La couleur est bleue',
    'vert' => 'La couleur est verte',
    default => 'Couleur inconnue',
};
?>
```

5. **Évaluation paresseuse (Lazy Evaluation)** :
   Les expressions conditionnelles avec `&&` et `||` en PHP supportent l'évaluation paresseuse, c'est-à-dire que PHP évalue les expressions de gauche à droite et s'arrête dès qu'il a trouvé le résultat.
   Cela peut être très utile pour optimiser les performances dans des conditions complexes.

```php
<?php
$age = 20;
$is_adult = $age >= 18 && ($age < 100);  // Si $age >= 18 est false, la deuxième condition n'est même pas évaluée.
?>
```

Les boucles

Les boucles en PHP sont utilisées pour répéter un bloc de code plusieurs fois. Il existe plusieurs types de boucles en PHP, notamment la boucle for, la boucle while et la boucle foreach.

for et foreach

La boucle for est utilisée pour répéter un bloc de code un nombre déterminé de fois.
Voici un code qui affiche les nombres de 0 à 4 :

```php
<?php
  for ($i = 0; $i < 5; $i++) {
    echo $i . ' ';
```

```
  }
?>
```

Il est également possible d'itérer sur un tableau grâce à la fonction `count()` ou `sizeof()` (un alias de `count()`)...

```php
<?php
$array = ["apple", "banana", "cherry"];

for ($i = 0; $i < count($array); $i++) {
    echo $array[$i] . "\n";
}
?>
```

...ou sur une chaîne de caractères grâce à la fonction built-in `strlen()` !

```php
$string = "hello";

for ($i = 0; $i < strlen($string); $i++) {
    echo $string[$i] . "\n";
}
```

La structure `foreach` est souvent plus simple et plus lisible lorsqu'on itère sur des tableaux, car elle évite d'avoir à gérer manuellement les indices. Chaque élément du tableau est directement assigné à la variable `$value` à chaque itération.

```php
<?php
$array = ["apple", "banana", "cherry"];

foreach ($array as $value) {
    echo $value . "\n";
}

# Il est toujours possible d'accéder à l'index avec foreach !
foreach ($fruits as $index => $fruit) {
    echo "Index: $index, Fruit: $fruit" . PHP_EOL;
}
?>
```

La dernière syntaxe dans cet exemple permet donc de travailler à la fois sur la valeur de l'élément et sur son index.

Bien qu'il ne soit pas aussi courant d'itérer directement sur une chaîne de caractères avec `foreach`, on peut facilement transformer une chaîne en un tableau de caractères avec `str_split()`, puis itérer dessus :

```php
<?php
$string = "hello";

foreach (str_split($string) as $char) {
    echo $char . "\n";
}
?>
```

while et do... while

La boucle while est utilisée pour répéter un bloc de code tant qu'une condition est vraie. Tant que la condition du `while` reste vraie, le bloc de code associé sera exécuté à chaque itération. Dès que la condition devient fausse, la boucle s'arrête.

```php
<?php
$i = 0;
    while ($i < 5) {
    echo $i++ . ' ';
}
?>
```

La boucle `do while` est similaire à la boucle `while`, mais avec une différence majeure : **la condition est vérifiée après l'exécution du bloc de code**. Cela signifie que le code à l'intérieur de la boucle est exécuté au moins **une fois**, même si la condition est **fausse** dès le début.

```php
<?php
$i = 0;
do {
    echo ++$i . ' ';
} while ($i < 5);
?>
```

Découverte des Tableaux

Les tableaux en PHP sont des structures de données qui permettent de stocker plusieurs valeurs dans une seule variable. Il existe deux types de tableaux en PHP : les tableaux indexés et les tableaux associatifs.

Tableaux indexés

Les tableaux indexés sont des tableaux dont les clés sont des nombres entiers.

```php
<?php
  $fruits = array('pomme', 'poire', 'abricot');
  // ou
  $fruits = ['pomme', 'poire', 'abricot'];
  echo $fruits[0]; // affiche 'pomme'
?>
```

Tableaux associatifs

Les tableaux associatifs sont des tableaux dont les clés sont des chaînes de caractères.

```php
<?php
  $personne = array('nom' => 'John Doe', 'age' => 30);
  // ou
  $personne = ['nom' => 'John Doe', 'age' => 30];
  echo $personne['nom']; // affiche 'John Doe'
?>
```

En PHP, tous les tableaux sont techniquement des **tableaux associatifs**. La seule différence réside dans la manière dont les clés sont générées ou attribuées (automatiquement ou manuellement).

En effet, lorsque vous créez un tableau indexé, PHP s'occupe d'attribuer des clés numériques automatiquement, ce qui évite d'écrire du code comme ceci :

```php
<?php
  $fruits = [0 => 'pomme', 1 => 'poire', 2 => 'abricot'];
?>
```

Ce qui revient à créer un tableau associatif avec des clés numériques. On peut donc dire qu'il est possible de mixer les deux syntaxes, car en réalité, tous les tableaux en PHP sont des

tableaux associatifs (ils possèdent toujours des paires clé/valeur).

```php
<?php
$personne = [
  'user_id67',          // clé: 0
  'nom' => 'John Doe', // clé: 'nom'
  'ROLE_ADMIN',         // clé: 1
  'age' => 30,          // clé: 'age'
];
?>
```

Bien que cette syntaxe ne soit pas recommandé, elle fonctionne. elle permet de bien comprendre la notion de clé implicite/clé explicite.

Découverte des Fonctions

Une **fonction** est un bloc de code réutilisable qui effectue une tâche précise. Elle permet d'**éviter la répétition**, d'**organiser le code** en unités logiques et de le **rendre plus lisible et maintenable**.

Définir une fonction

En PHP, vous pouvez créer vos propres fonctions avec le mot-clé `function`. Une fonction permet de regrouper un bloc de code que vous pouvez appeler plusieurs fois.

```php
<?php
function greet() {
  echo "Bonjour !";
}
greet(); // affiche "Bonjour !"
?>
```

Fonction avec paramètre(s)

Une fonction peut recevoir un ou plusieurs paramètres. Ceux-ci permettent de rendre la fonction plus flexible.

```php
<?php
function greet($firstName, $lastName)
{
  echo "Bonjour $firstName ", strtoupper($lastName), " !";
}
greet("Alice", "Dupont"); // affiche "Bonjour Alice DUPONT !"
?>
```

A partir de PHP 8, il est possible d'utiliser les arguments nommés. Cela permet de passer les arguments à une fonction **sans respecter l'ordre défini dans la déclaration**, tant que vous précisez le nom de chaque paramètre.

```php
<?php
function greet($firstName, $lastName) {
  echo "Bonjour $firstName ", strtoupper($lastName), " !";
}

// Appel avec des arguments nommés (ordre inversé)
greet(lastName: "Dupont", firstName: "Alice"); // affiche "Bonjour Alice
DUPONT !"
?>
```

Valeur de retour

Une fonction peut retourner une valeur avec `return`. Cela permet de récupérer un résultat et de le manipuler.

```php
<?php
function getGreeting($name) {
  return "Bonjour $name !";
}

$message = getGreeting("Alice");
echo $message; // affiche "Bonjour Alice !"
?>
```

Paramètre par défaut

Vous pouvez définir une valeur par défaut à un paramètre. Elle sera utilisée si aucun argument n'est fourni.

```php
<?php
function greet($name = "inconnu") {
  echo "Bonjour $name !";
}
greet();          // affiche "Bonjour inconnu !"
greet("Alice");   // affiche "Bonjour Alice !"
?>
```

Passage par référence

Si vous souhaitez modifier une variable en dehors de la fonction, vous pouvez passer le paramètre par référence en utilisant `&`.

```php
<?php
function addExclamation(&$text) {
  $text .= " !";
}
$message = "Bonjour";
addExclamation($message);
echo $message; // affiche "Bonjour !"
?>
```

Fonctions anonymes (closures)

Il est possible de stocker une fonction dans une variable. Cela peut servir notamment comme fonction de rappel (callback).

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

echo $greet("Alice"); // affiche "Bonjour Alice !"
?>
```

Fonctions fléchées (arrow functions)

Depuis PHP 7.4, vous pouvez utiliser une syntaxe plus courte pour écrire des fonctions anonymes simples.

```php
<?php
$greet = fn($name) => "Bonjour $name !";
echo $greet("Alice"); // affiche "Bonjour Alice !"
?>
```

Accès à une variable globale

En PHP, une variable définie en dehors d'une fonction **n'est pas accessible directement** à l'intérieur de celle-ci. Pour y accéder, il faut utiliser le mot-clé `global`.
Attention : **l'utilisation abusive de `global` est déconseillée** ! Cela rend le code moins lisible, plus difficile à maintenir, et crée des dépendances cachées.

```php
<?php
$message = "Bonjour tout le monde !";

function displayMessage() {
  global $message;
  echo $message;
}

displayMessage(); // affiche "Bonjour tout le monde !"
?>
```

PHP fournit un tableau associatif spécial nommé `$GLOBALS`, qui contient toutes les variables globales accessibles dans le script. Cela permet d'y accéder sans utiliser le mot-clé `global`.
A utiliser avec modération !

```php
<?php
$message = "Bonjour tout le monde !";

function displayMessage() {
  echo $GLOBALS['message'];
}

displayMessage(); // affiche "Bonjour tout le monde !"
?>
```

La meilleure pratique consiste à **passer la variable en paramètre** à la fonction.

```php
<?php
$message = "Bonjour tout le monde !";
```

```php
function displayMessage($text) {
  echo $text;
}

displayMessage($message); // affiche "Bonjour tout le monde !"
?>
```