Les Bases du langage PHP

Une introduction au PHP

Mots clé : PHP, MySQL

DWWM - PHP - 010724

- JD.TOULOUSE -

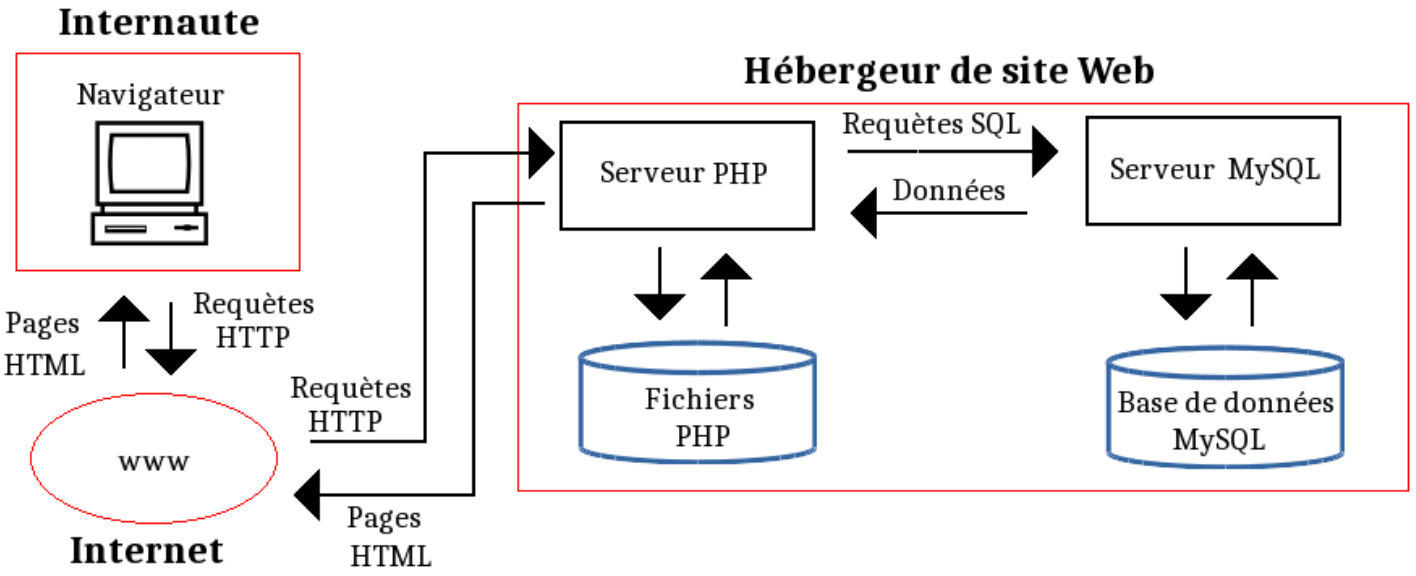
PHP

1. Introduction au langage PHP

2. Histoire de PHP

PHP (Hypertext Preprocessor) a été créé par Rasmus Lerdorf en 1994. À l'origine, il s'agissait d'un ensemble de scripts CGI (Common Gateway Interface) utilisés pour le suivi des visiteurs sur son site web personnel. Au fil du temps, PHP a évolué pour devenir un langage de programmation complet, capable de gérer des tâches complexes et de développer des applications web dynamiques.

3. Environnement d'exécution & Outils



3.1. XAMPP et autres outils

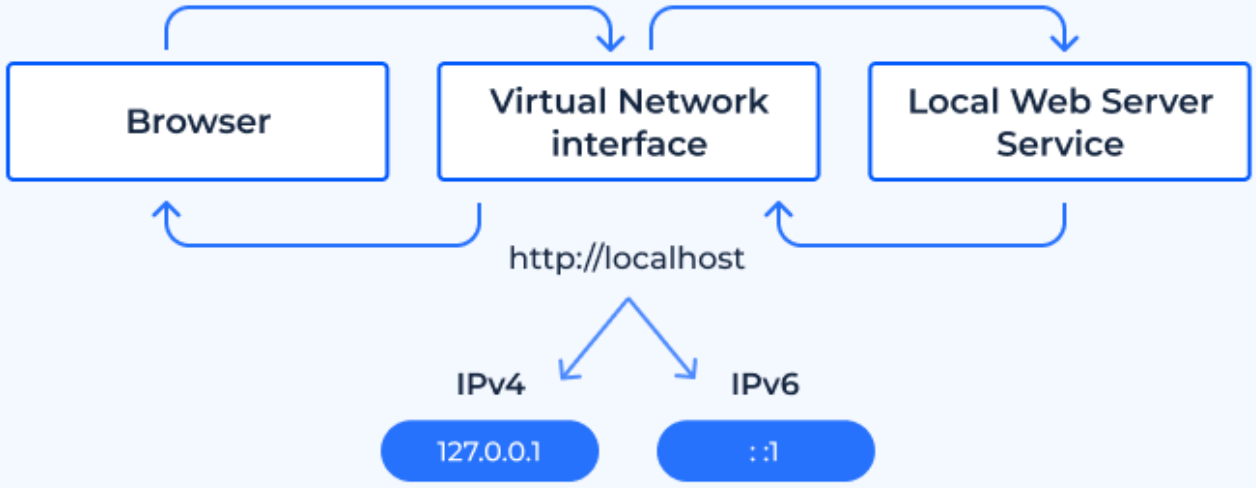
> **LAMP**, **MAMP**, **XAMPP**, **WAMP**, **LARAGON** sont des logiciels qui permettent de créer un serveur web local sur votre ordinateur.
>
> XAMPP est un logiciel libre et open-source qui facilite le déploiement d'un environnement de développement web local. Il est souvent utilisé pour créer et tester des sites web dynamiques avant de les mettre en ligne.
>
> `XAMPP` est un acronyme :

1. **X**: signifie"multiplateforme". XAMPP est utulisable sur plusieurs systèmes d'exploitation (Windows, Linux et macOS).
2. **A**: Apache = serveur web inclus dans XAMPP qui permet de servir des fichiers web et de gérer les requêtes HTTP.
3. **M**: MySQL (ou parfois MariaDB) = système de gestion de base de données relationnelle utilisé dans XAMPP.
4. **P**: PHP = langage de programmation côté serveur utilisé pour créer des applications web dynamiques.
5. **P**: Perl = langage de programmation polyvalent souvent utilisé pour écrire des scripts CGI pour le web.

XAMPP installe également **phpMyAdmin**, une application web qui gère les bases de données MySQL à travers une interface graphique.

3.2. Localhost

Localhost est un **nom d'hôte réservé** qui pointe vers l'**adresse IP** spéciale **127.0.0.1** (IPv4) ou **::1** (IPv6) sur votre ordinateur.



Les trois notations, à savoir `localhost`, `127.0.0.1` et `http://[::1]`, font référence au même emplacement, à savoir l'Interface Réseau Virtuelle de l'ordinateur.

3.3. Serveur local

Contrairement à un serveur en ligne accessible depuis Internet, un serveur de développement local reste sur votre machine, créant un espace de travail isolé pour le développement, les tests et le débogage de vos projets.

> Pour tirer pleinement parti de l'environnement de développement local, les serveurs locaux utilisent `localhost` comme **point d'accès** à leurs services.

4. Documentation

- https://www.php.net/manual/fr/
- https://devdocs.io/php/

5. Visual Studio Code (extensions)

- Code Runner
- PHP DevSense
- PHP IntelliSense
- PHP SERVER

6. Balises php

> En PHP, `<?` est utilisé comme une balise d'ouverture pour délimiter le code PHP dans un fichier.
> C'est une forme abrégée de `<?php`, qui est l'autre balise couramment utilisée.
> Ces balises indiquent au serveur web que le code entre elles doit être interprété en tant que code PHP.
> ***Exemple :***

```php
<?php

    echo "Bonjour!";
?>
```

En PHP, la balise de fermeture `?>` n'est pas obligatoire à la fin d'un fichier. Dans certains cas, il est même recommandé de ne pas utiliser la balise de fermeture `?>` à la fin d'un fichier PHP.
La raison principale est d'éviter l'inclusion accidentelle d'espaces ou de caractères indésirables après la balise de fermeture `?>`, ce qui pourrait causer des erreurs indésirables, notamment des problèmes de sortie indésirable (output) avant l'envoi des en-têtes HTTP.
Il est recommandé de laisser le fichier PHP se terminer simplement après la dernière instruction PHP sans utiliser explicitement la balise de fermeture `?>`. Cela garantit que le fichier reste purement du

code PHP sans risque d'espaces ou de caractères supplémentaires indésirables.

Exemple :

```php
<?php
echo "Bonjour!";
```

7. echo

L'instruction `echo` est utilisée pour afficher du texte ou des variables dans PHP.

Exemple :

```php
echo "Bonjour";
echo 'Bonjour';
echo ("Bonjour");
echo ('Bonjour');
echo "Le language \"PHP\" est très agréable à appréhender.";
echo 'Aujourd\'hui il fait très beau.';
echo 'Il fait', ' très beau ', 'ce matin.', '<br>';
echo "Il fait" . " très beau ce matin !";
```

7.1. nl2br()

La fonction `nl2br` en PHP est utilisée pour convertir les sauts de ligne (`\n` ou `\r\n`) en balises HTML `<br>` dans une chaîne de caractères.

Cela permet d'afficher du texte avec des sauts de ligne correctement formatés dans une page web.

\r (retour chariot) :

- Le retour chariot, abrégé en "\r", est un caractère de contrôle utilisé pour déplacer le curseur d'impression ou le point d'insertion de texte à la position initiale de la ligne actuelle, généralement le début de la ligne.
- Il est souvent utilisé en conjonction avec la nouvelle ligne ("\n") dans les systèmes de type Windows pour représenter un saut de ligne complet ("\r\n").

  \n (nouvelle ligne) :
- La nouvelle ligne, abrégée en "\n", est un caractère de contrôle qui indique la fin d'une ligne de texte. Il est utilisé pour passer à la ligne suivante dans un document ou un fichier texte.

> - "\n" est couramment utilisé dans les systèmes Unix et Linux ainsi que dans de nombreux langages de programmation pour représenter un saut de ligne.
> - Contrairement à "\r", "\n" ne revient pas au début de la ligne ; il ne fait que passer à la ligne suivante.
>
> **nl2br() :**

```php
echo nl2br("Bonjour\n");
echo nl2br("Bonsoir\n");
$texte = "Ceci est une phrase avec\n un saut de ligne.";
$texte_formate = nl2br($texte);
echo $texte_formate;
```

7.2. PHP_EOL

- `PHP_EOL` est une autre fonctionnalité importante de PHP liée à la manipulation des chaînes de caractères, surtout dans le contexte des sauts de ligne.
- représente le 'End Of Line' (fin de ligne) pour la plate-forme sur laquelle le script est en train de s'exécuter.
- constante pré-définie en PHP qui est utilisée pour insérer un saut de ligne de manière correcte et indépendante du système d'exploitation.
- Son utilisation est particulièrement pertinente dans les scripts conçus pour être exécutés sur différents systèmes d'exploitation.
- **Non recommandé pour les sorties HTML :** Il est important de noter que `PHP_EOL` n'est pas utilisé pour les sauts de ligne dans le HTML. Pour cela, la balise `<br>` est appropriée.

  Exemple :

```php
echo "Première ligne" . PHP_EOL;
echo "Deuxième ligne" . PHP_EOL;
```

7.3. Heredoc

- `Heredoc` est une méthode de déclaration de chaînes de texte longues sans avoir besoin de concaténer ou d'échapper manuellement les caractères.
- Il est très utile pour définir des chaînes HTML ou SQL complexes.

- La syntaxe commence par `<<<` suivi d'un identifiant, puis la chaîne, et se termine par le même identifiant.
  Exemple :

```
code$str = <<<EOD
Exemple de chaîne
sur plusieurs lignes
utilisant la syntaxe heredoc.
EOD;
```

8. Variables

8.1. Règle de nommage

Pour écrire un nom de variable, tous les programmeurs doivent suivre des règles prédéfinies dans chaque langage de programmation. Différents types de règles définies dans différents langages de programmation :

- Un nom de variable doit être suivi d'un symbole `$`.
- Un nom de variable doit commencer par des lettres de l'alphabet ou un trait de soulignement (_).
- Un nom de variable ne doit contenir aucun symbole, à l'exception du trait de soulignement.
- Un nom de variable peut être une combinaison de lettres et de chiffres (aucun chiffre au début du nom).
- Un nom de variable ne peut pas contenir d'espaces.
- Les variables sont sensibles à la casse (les noms de variables `oui` et `OUI` seront considérées comme différents).

8.2. Déclaration

```
$age = 25;
$nom = "John";
$est_majeur = true;
$taille = 1.75;
$an = "an";
```

8.3. Concaténation

```
echo "Bonjour " . $nom . ", vous avez " . $age . " ans.";
```

8.4. Affichage

```
echo $age;
echo $nom;
echo $est_majeur;
echo $taille;
```

Concaténation avec guillemets doubles + balise HTML :

```
echo "<p>Bonjour " . $nom . ", vous avez " . $age . " ans.</p>";
```

Concaténation avec guillemets simples + balise HTML :

```
echo '<p>Bonjour ' . $nom . ', vous avez ' . $age . ' ans.</p>';
```

Guillemets doubles + balise HTML :

```
echo "<p>Bonjour $nom, vous avez $age ans.</p>";
```

Guillemets doubles + balise HTML + {}

```
echo "<p>Bonjour {$nom}, vous avez {$age} ${an}s.</p>";
```

8.5. Destruction

```
<?php
$couleur = 'rouge';
echo $rouge;
unset($couleur);
```

8.6. Variables superglobales

PHP fournit un très grand nombre de variables prédéfinies accessibles à tous vos scripts (on parle alors de ***variables superglobales***), ce qui signifie qu'elles sont disponibles quelque soit le contexte du script. Les variables superglobales sont des tableaux.

Une variable superglobale est une variable qui est **toujours accessible**, quel que soit le contexte (à l'intérieur d'une fonction, d'une classe, etc.).

Une variable superglobale n'est pas forcément une variable d'environnement.

Nom de la variable prédéfinie	Description	Superglobale ?
$GLOBALS	Référence toutes les variables disponibles dans un contexte global	OUI
$_SERVER	Variables de serveur et d'exécution	OUI
$_GET	Variables HTTP GET	OUI
$_POST	Variables HTTP POST	OUI
$_FILES	Variable de téléchargement de fichier via HTTP	OUI
$_REQUEST	Variables de requête HTTP	OUI
$_SESSION	Variables de session	OUI
$_ENV	Variables d'environnement	OUI
$_COOKIE	Cookies HTTP	OUI
$http_response_header	En-têtes de réponse HTTP	NON
$argc	Le nombre d'arguments passés au script	NON
$argv	Tableau d'arguments passés au script	NON

8.7. Variables d'environnement

Une variable d'environnement est une variable qui est **définie par le serveur**. Une variable d'environnement est une variable superglobale.

Elle contient des informations sur le serveur et sur le client. Les variables d'environnement sont des tableaux

Les **variables d'environnement** sont très utiles en programmation (notamment en PHP). Elles permettent d'avoir une idée claire sur l'environnement du programme qui peut être soit le client ou le serveur. Elles peuvent ainsi fournir des informations sur le type du serveur, son adresse IP, navigateur du visiteur, le nom de sa machine...

Variable environnement	Description
$_SERVER['DOCUMENT_ROOT']	Racine du serveur
$_SERVER['HTTP_ACCEPT_LANGUAGE']	Langage accepté par le navigateur
$_SERVER['HTTP_HOST']	Nom de domaine du serveur
$_SERVER['HTTP_USER_AGENT']	Type de navigateur
$_SERVER['PATH_INFO']	Chemin WEB du script
$_SERVER['PATH_TRANSLATED']	Chemin complet du script
$_SERVER['REQUEST_URI']	Chemin du script
$_SERVER['REMOTE_ADDR']	Adresse IP du client
$_SERVER['REMOTE_PORT']	Port de la requête HTTP
$_SERVER['QUERY_STRING']	Liste des paramètres passés au script
$_SERVER['SERVER_ADDR']	Adresse IP du serveur
$_SERVER['SERVER_ADMIN']	Adresse de l'administrateur du serveur
$_SERVER['SERVER_NAME']	Nom local du serveur
$_SERVER['SERVER_SIGNATURE']	Type de serveur
$_SERVER['REQUEST_METHOD']	Méthode d'appel du script

9. Constantes

> Une constante est un identifiant (un nom) qui représente une valeur unique. Comme son nom l'indique, cette valeur ne peut pas changer durant l'exécution du script (d'où son nom).
> Une constante est définie avec la fonction `define()`. Elle prend deux paramètres : le nom de la constante et sa valeur : `define("NOM", "valeur");`
> Les constantes :
>
> - sont généralement écrites en majuscules.
> - sont utiles pour stocker des informations qui ne doivent pas être modifiées durant l'exécution du script.
> - sont globales : elles sont accessibles partout dans le script, y compris dans les fonctions.
> - ne peuvent pas être redéfinies : une fois qu'une constante est définie, elle ne peut plus être modifiée ou supprimée.
> - sont automatiquement converties en chaînes de caractères lorsqu'elles sont affichées avec echo ou print.
> - sont automatiquement converties en nombres lorsqu'elles sont utilisées dans une opération mathématique.

```php
define("NOM", "John");
echo NOM;
```

9.1. Constantes pré-définies

```php
echo PHP_VERSION;
echo PHP_OS;
echo PHP_INT_MAX;
echo PHP_INT_MIN;
echo PHP_INT_SIZE;
echo PHP_FLOAT_MAX;
echo PHP_FLOAT_MIN;
echo PHP_FLOAT_DIG;
echo PHP_FLOAT_EPSILON;
echo PHP_FLOAT_MIN;
```

9.2. Constantes magiques pré-définies

```php
echo __LINE__;
echo __FILE__;
echo __DIR__;
echo __FUNCTION__;
echo __CLASS__;
echo __METHOD__;
echo __NAMESPACE__;
echo __TRAIT__;
echo __NAMESPACE__;
```

10. Types de données

10.1. Vue générale

Type de Données	Description	Exemple
Entier (int)	Nombres entiers, positifs ou négatifs, sans décimales.	`$nombreEntier = 42;`
Nombre à Virgule Flottante (float)	Nombres décimaux avec une virgule flottante.	`$prix = 19.99;`
Chaîne de Caractères (string)	Séquence de caractères, encadrée par des guillemets simples ou doubles.	`$texte = "Bonjour, monde!";`
Booléen (bool)	Valeur de vérité, `true` (vrai) ou `false` (faux).	`$estVrai = true;`
Tableau (array)	Collection de valeurs indexées par des clés (noms ou indices).	`$tableau = [1, 2, 3, 4];`
Objet (object)	Instance d'une classe avec propriétés et méthodes.	Voir exemple dans le script précédent
NULL	Variable sans valeur ou non initialisée.	`$nulleVariable = null;`

10.2. Type de données primitifs

> Les types de données primitifs sont les plus fondamentaux en PHP et sont utilisés pour stocker **des valeurs de base**.
>
> Ils sont souvent utilisés dans les opérations mathématiques, la manipulation de texte, la logique de contrôle, etc.

Type de Données	Description	Exemple
Entier (int)	Nombres entiers, positifs ou négatifs, sans décimales.	`$nombreEntier = 42;`
Float (nombre à virgule flottante)	Nombres décimaux avec une virgule flottante.	`$prix = 19.99;`
Chaîne de Caractères (string)	Séquence de caractères, encadrée par des guillemets simples ou doubles.	`$texte = "Bonjour, monde!";`
Booléen (bool)	Valeur de vérité, `true` (vrai) ou `false` (faux).	`$estVrai = true;`
NULL	Variable sans valeur ou non initialisée.	`$nulleVariable = null;`

10.3. Type de données composées

Les types de données composées en PHP sont utilisés pour stocker des **ensembles de données plus complexes et structurées**. Les tableaux sont particulièrement polyvalents et peuvent contenir une variété de valeurs, tandis que les objets permettent de créer des structures de données personnalisées avec des propriétés et des méthodes.

Les ressources sont utilisées pour gérer des ressources externes, telles que des fichiers ou des connexions de base de données.

Type de Données Composées	Description	Exemples
Tableau (array)	Collection de valeurs indexées par des clés (noms ou indices).	`$tableau = [1, 2, 3, 4];`
Objet (object)	Instance d'une classe avec propriétés et méthodes.	Voir module PHP POO
Ressource (resource)	Représente une ressource externe, telle qu'une connexion de base de données.	-

10.4. gettype() et settype()

> `gettype()` et `settype()` sont deux fonctions en PHP qui permettent de gérer les types de données des variables

- **gettype()** : permet de récupérer le type de données actuel d'une variable. Elle prend en argument la variable dont vous voulez connaître le type et renvoie une chaîne de caractères représentant le type de données.

```php
$nombre = 42;
$type = gettype($nombre);
```

- **settype()** : Cette fonction permet de changer le type de données d'une variable. Elle prend deux arguments : la variable que vous souhaitez modifier et le nouveau type que vous voulez lui attribuer.

```php
$nombre = "42";
settype($nombre, "integer");
```

11. Conversion de types

> La conversion de types est couramment utilisée en PHP pour **modifier le type d'une variable ou d'une valeur** d'une manière spécifique. Il est important de noter que la conversion de type **peut entraîner une perte de données** si la conversion n'est pas possible ou appropriée.
>
> Il est donc essentiel de comprendre les différents mécanismes de conversion de types disponibles en PHP et de les utiliser avec précaution en fonction des besoins du programme.

Fonction/Opérateur	Description	Exemple
(type)	Opérateur de conversion explicite. Permet de forcer la conversion d'un type à un autre.	`(int) 42.75;` renvoie 42
`intval()`	Fonction pour convertir en entier.	`intval("123");` renvoie 123
`floatval()`	Fonction pour convertir en nombre à virgule flottante.	`floatval("3.14");` renvoie 3.14
`strval()`	Fonction pour convertir en chaîne de caractères.	`strval(42);` renvoie "42"
`boolval()`	Fonction pour convertir en booléen.	`boolval(0);` renvoie false
`(bool)`	Conversion en booléen à l'aide de l'opérateur de conversion explicite.	`(bool) "true";` renvoie true
`json_decode()`	Fonction pour convertir une chaîne JSON en une structure de données PHP.	`$donnees = json_decode('{"nom": "Alice"}');`
`(array)`	Conversion en tableau à l'aide de l'opérateur de conversion explicite.	`(array) $objet;` convertit un objet en tableau
`(object)`	Conversion en objet à l'aide de l'opérateur de conversion explicite.	`(object) $tableau;` convertit un tableau

11.1. Conversion implicite (juggling)

> La conversion implicite, également appelée "**coercition de type**" ou "**juggling**", se produit automatiquement lorsqu'une opération ou une expression en PHP nécessite des types de données différents. Dans de telles situations, PHP tente de convertir les types de données de manière transparente pour que l'opération puisse être effectuée.
> ***Exemple :***

```php
<?php
$chaine = "42";
$nombre = $chaine + 10;
echo "Conversion de chaîne en nombre : $nombre\n";
$nombre = 42;
$chaine = "Le nombre est " . $nombre;
echo "Conversion de nombre en chaîne : $chaine\n";
$vrai = true;
$faux = false;
$vrai_entier = $vrai + 10;
$faux_entier = $faux + 10;
echo "Conversion de booléens en nombres : vrai_entier = $vrai_entier, faux_entier = $faux_entier\n";
$entier = 42;
$flottant = $entier / 2;
echo "Conversion entre types numériques : flottant = $flottant\n";
$chaine_non_vide = "Bonjour";
$chaine_vide = "";
$est_vrai = (bool)$chaine_non_vide;
$est_faux = (bool)$chaine_vide;
echo "Conversion de chaînes en booléens : est_vrai = $est_vrai, est_faux = $est_faux\n";
?>
```

11.2. Conversion explicite

> La conversion explicite, également appelée "**casting**", se produit lorsque vous spécifiez explicitement le type de données vers lequel vous voulez convertir une valeur. Cela se fait généralement à l'aide d'opérateurs ou de fonctions de conversion spécifiques en PHP.
> ***Exemple :***

```php
<?php
$nombre = 42.75;
$entier = (int)$nombre;
echo "Conversion en entier : $entier\n";
$chaine = "3.14";
$flottant = (float)$chaine;
echo "Conversion en flottant : $flottant\n";
$nombre = 42;
$chaine = strval($nombre);
echo "Conversion en chaîne : $chaine\n";
$valeur = 0;
$booleen = (bool)$valeur;
echo "Conversion en booléen : " . ($booleen ? "true" : "false") . "\n";
?>
```

11.3. Fonctions de conversion explicite de types

> En PHP, il existe plusieurs fonctions pour effectuer des conversions explicites de type.

```php
<?php
$chaine = "42";
$entier = intval($chaine);
echo "Conversion en entier : $entier\n";
$chaine = "3.14";
$flottant = floatval($chaine);
echo "Conversion en flottant : $flottant\n";
$nombre = 42;
$chaine = strval($nombre);
```

```
echo "Conversion en chaîne : $chaine\n";
?>
```

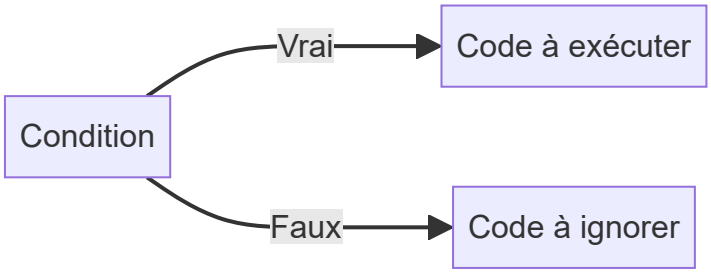
12. Structures conditionnelles (structures de contrôle)

> Une structure conditionnelle, en programmation, est une construction qui permet d'exécuter différents blocs de code en fonction de l'évaluation d'une ou plusieurs conditions. Ces structures sont essentielles pour prendre des décisions dans un programme en fonction de situations différentes.

- **Condition** : Une structure conditionnelle évalue une expression ou une condition logique. Cette condition peut être vraie (true) ou fausse (false).
- **Bloc de code** : Un bloc de code est un ensemble d'instructions qui est exécuté si la condition de la structure conditionnelle est vraie.
- **Exécution alternative** : Dans certaines structures conditionnelles, il est possible de spécifier un bloc de code alternatif à exécuter si la condition est fausse.

12.1. Structure de type `if`

> `if` permet d'exécuter un bloc de code si une condition est vraie.

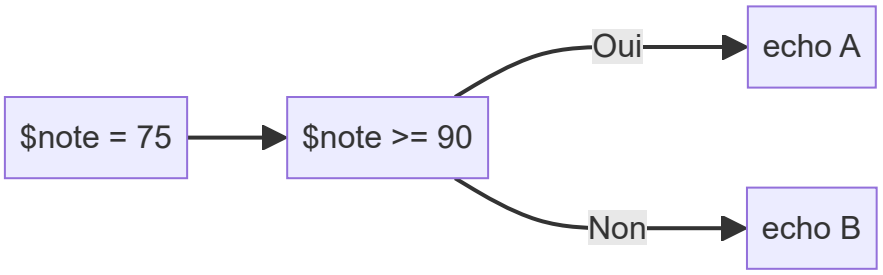


Exemple :

```
$note = 85;
if ($note >= 90) {
    echo "A";
}
```

12.2. Structure de type `if ... else`

> `if` permet d'exécuter un bloc de code si une condition est vraie.
> Elle peut être suivie d'une clause `else` pour exécuter un bloc de code alternatif si la condition est fausse.

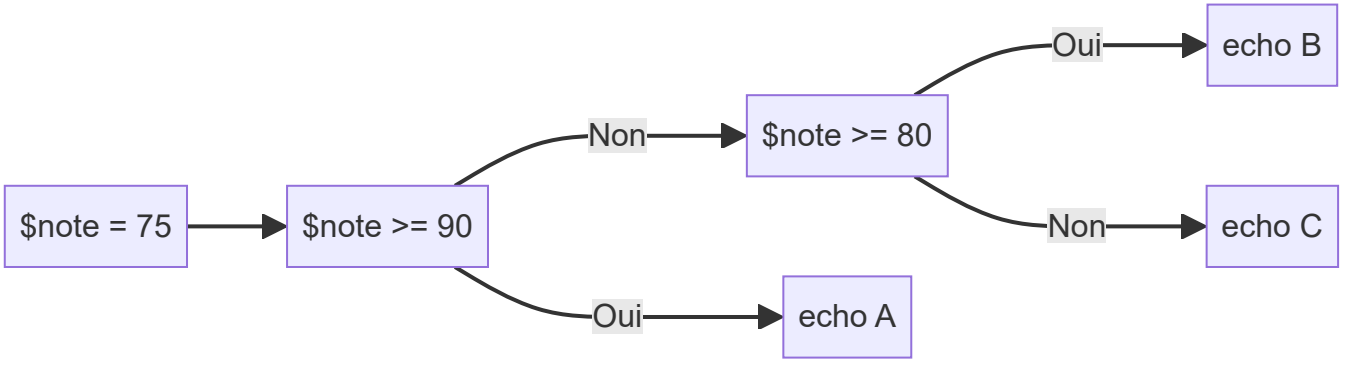


Exemple :

```php
$note = 75;
if ($note >= 90) {
    echo "A";
} else {
    echo "B";
}
```

12.3. Structure if...elseif...else

> La structure `elseif` (ou `else if`) est utilisée pour évaluer plusieurs conditions en cascade.
> Elle suit généralement une instruction `if` et permet de tester une condition différente si la première condition n'est pas vraie.

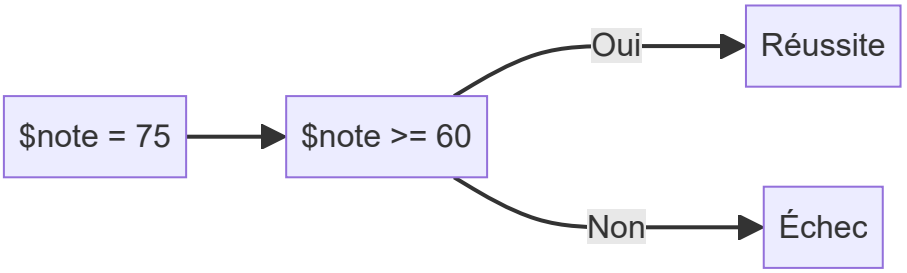


Exemple :

```php
$note = 75;
if ($note >= 90) {
    echo "A";
} elseif ($note >= 80) {
    echo "B";
} else {
    echo "C";
}
```

12.4. Condition ternaire

> L'opérateur ternaire, également appelé opérateur conditionnel, est une alternative concise à la structure `if...else` pour évaluer une expression conditionnelle et renvoyer une valeur en fonction de la condition.
> La syntaxe de l'opérateur ternaire est `condition ? valeur_si_vrai : valeur_si_faux`.



Exemple :

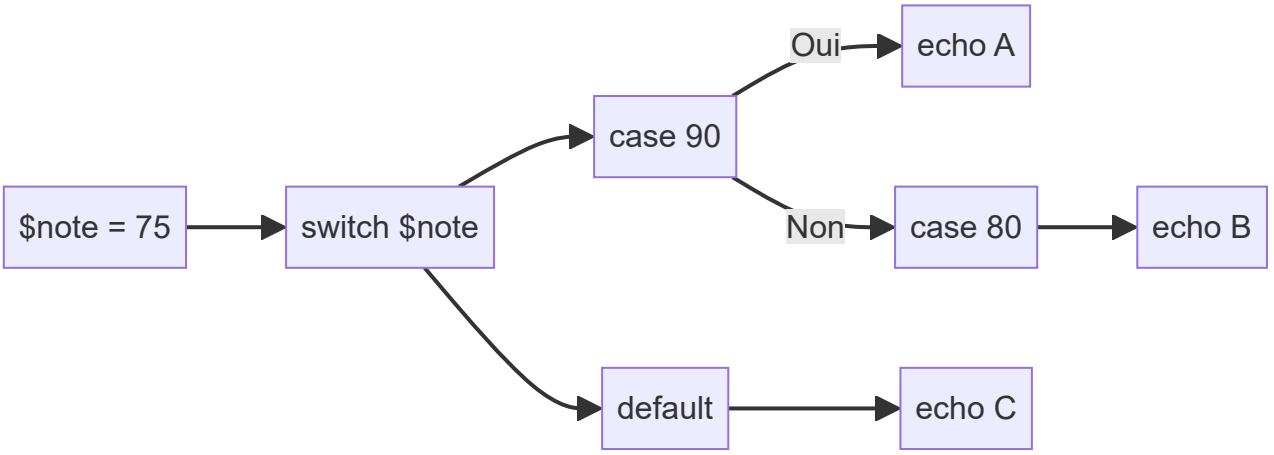
```php
$note = 75;
$resultat = ($note >= 60) ? "Réussite" : "Échec";
echo $resultat;
```

Autre forme d'écriture possible :

```php
$note = 75;
echo ($note >= 60) ? "Réussite" : "Échec";
```

12.5. Structure conditionnelle switch

> La structure de contrôle `switch` est utilisée dans de nombreux langages de programmation, dont PHP, pour gérer plusieurs cas en fonction de la valeur d'une expression.
>
> Elle offre une alternative élégante aux séries de conditions `if...elseif...else` lorsqu'il y a plusieurs options à considérer.

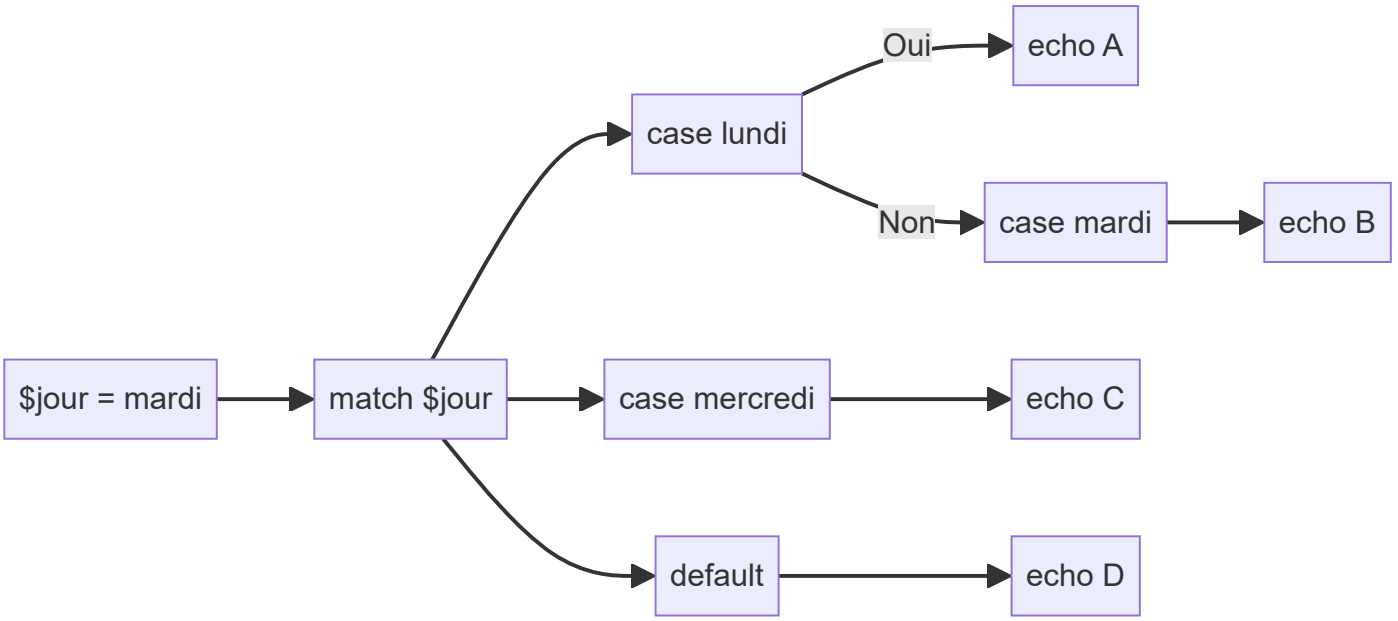


Exemple :

```php
$jour = "lundi";
switch ($jour) {
    case "lundi":
        echo "C'est le début de la semaine.";
        break;
    case "mardi":
        echo "C'est le deuxième jour de la semaine.";
        break;
    case "mercredi":
        echo "C'est le milieu de la semaine.";
        break;
    default:
        echo "C'est un autre jour de la semaine.";
}
```

12.6. Structure conditionnelle match

> L'expression `match`, également appelée l'instruction `match`, est une fonctionnalité introduite dans PHP 8. Elle permet de simplifier et d'améliorer les structures de contrôle conditionnelles pour comparer une valeur avec plusieurs cas et renvoyer un résultat en fonction de la correspondance.
>
> L'expression `match` est plus concise que `switch` et permet une syntaxe plus fluide pour évaluer des expressions et produire des résultats en fonction des valeurs correspondantes.



Exemple :

```php
$jour = "mardi";
$resultat = match ($jour) {
    "lundi" => "C'est le début de la semaine.",
    "mardi" => "C'est le deuxième jour de la semaine.",
    "mercredi" => "C'est le milieu de la semaine.",
    default => "C'est un autre jour de la semaine.",
};
echo $resultat;
```

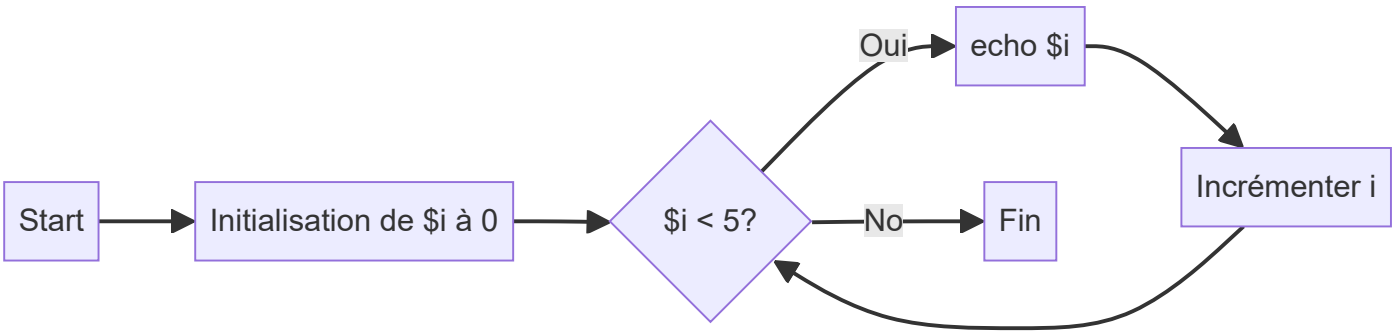
13. Boucles

> Les boucles contrôlent la manière dont le flux du programme est répété ou itéré. Elles permettant d'exécuter un bloc de code de manière **répétée** tant qu'une condition est satisfaite (vraie) ou pour un **nombre d'itérations spécifié**. Elles sont largement utilisées pour automatiser des tâches répétitives et traiter des données de manière itérative.
> Une boucle typique comprend les étapes suivants :

1. **Initialisation** : définir une variable de contrôle ou une valeur initiale qui sera utilisée dans la condition de la boucle.
2. **Condition** : Une expression logique est évaluée à chaque itération pour déterminer si la boucle doit continuer à s'exécuter. Si la condition est vraie, la boucle continue, sinon, elle se termine.
3. **Incrémentation ou Décrémentation** : modifier la variable de contrôle à chaque itération pour éventuellement atteindre la condition d'arrêt.

13.1. Boucle `for`

> La boucle `for` est idéale lorsque vous connaissez le nombre d'itérations à l'avance.
>
> Elle se compose de trois parties : l'initialisation, la condition de continuation et l'incrémentation.

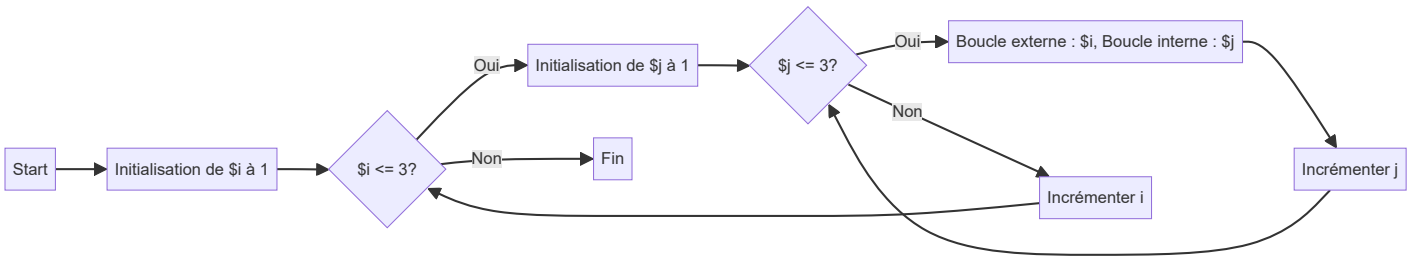


Exemple :

```php
for ($i = 0; $i < 5; $i++) {
    echo $i;
}
```

13.2. Boucle for imbriquées

> Les boucles imbriquées, également appelées boucles for imbriquées, sont des boucles placées à l'intérieur d'une autre boucle.
>
> Elles sont utilisées pour effectuer des itérations multiples et sont couramment utilisées pour parcourir des tableaux à deux dimensions (matrices), générer des combinaisons ou des permutations, ou pour effectuer des opérations sur des structures de données complexes.



Exemple :

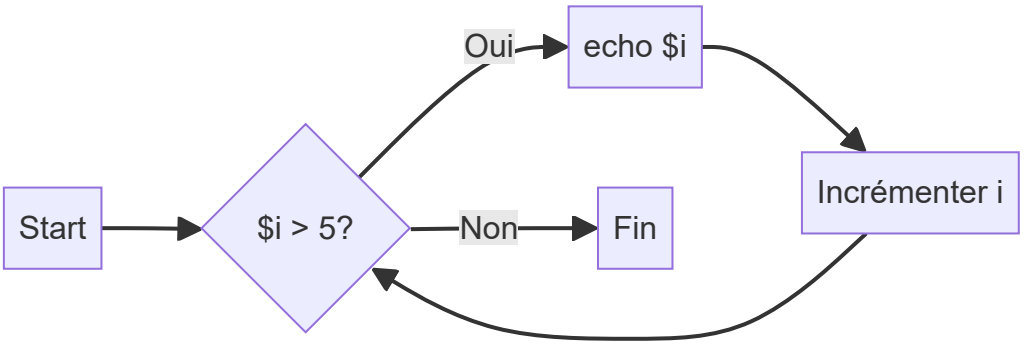
```php
for ($i = 1; $i <= 3; $i++) {
    for ($j = 1; $j <= 3; $j++) {
        echo "Boucle externe : $i, Boucle interne : $j<br>";
    }
}
```

Exemple :

```php
for ($i = 1; $i <= 10; $i++) {
    for ($j = 1; $j <= 10; $j++) {
        echo "$i x $j = " . ($i * $j) . " ";
    }
    echo "\n";
}
```

13.3. Boucle `while`

> La boucle `while` est utilisée lorsque vous ne connaissez pas le nombre d'itérations à l'avance, mais vous avez une condition à vérifier. Elle continue tant que la condition est vraie.

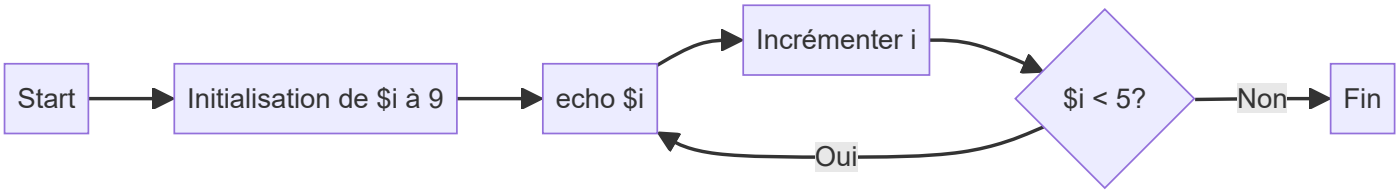


Exemple :

```php
$i = 9;
while ($i > 5) {
    echo $i;
    $i--;
}
```

13.4. Boucle `do...while`

> La boucle `do...while` est similaire à la boucle `while`, mais elle garantit au moins une exécution du code avant de vérifier la condition.

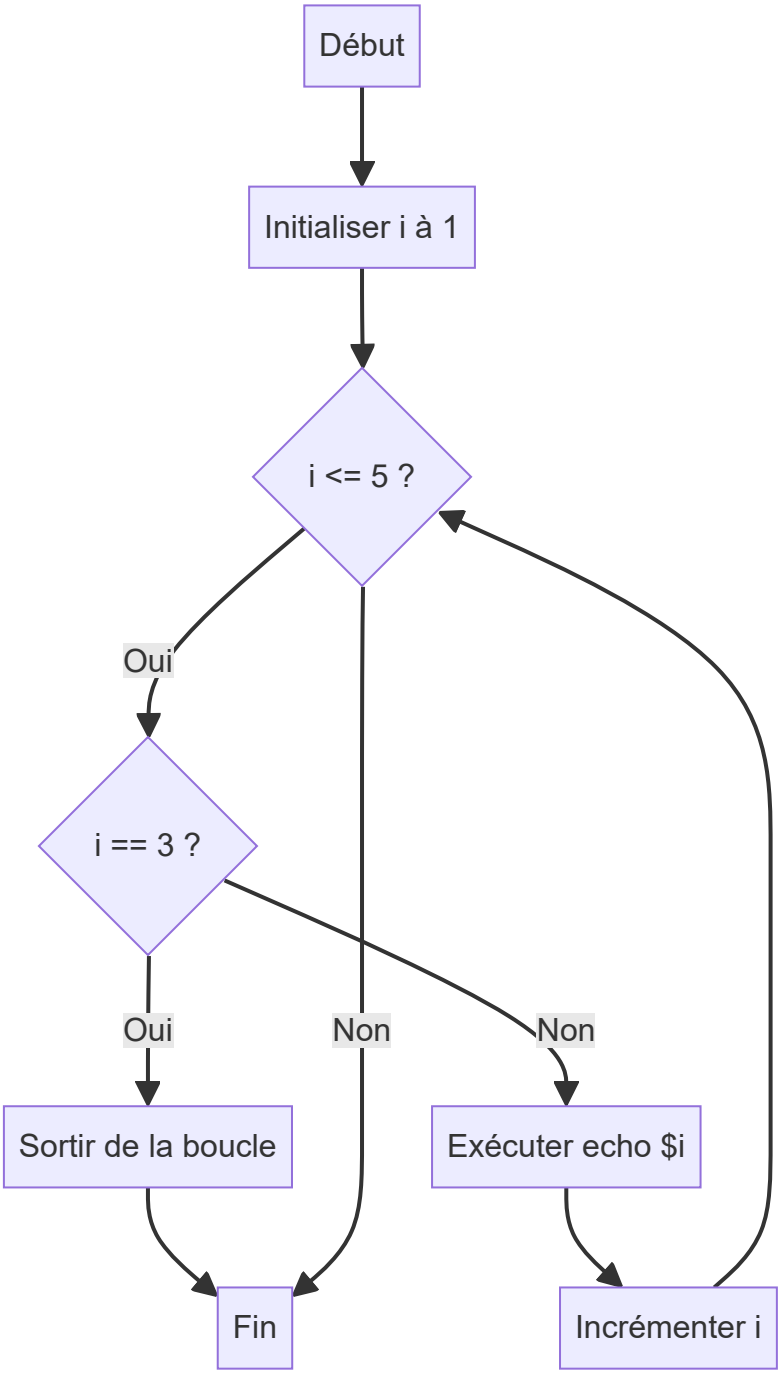


Exemple :

```php
$i = 9;
do {
    echo $i;
    $i--;
} while ($i < 5);
```

13.5. Instruction `break`

> L'instruction `break` est utilisée pour sortir prématurément d'une boucle lorsqu'une condition spécifique est remplie.
>
> Elle est couramment utilisée dans les boucles `for`, `while`, et `do...while`.



```php
for ($i = 1; $i <= 5; $i++) {
    if ($i == 3) {
        break;
    }
    echo $i . " ";
}
```

13.6. Instruction `continue`

> L'instruction `continue` est utilisée pour sauter à l'itération suivante d'une boucle sans exécuter le reste du code de l'itération actuelle.
>
> Elle est également couramment utilisée dans les boucles `for`, `while`, et `do...while`.

```php
for ($i = 1; $i <= 5; $i++) {
    if ($i == 3) {
        continue;
    }
    echo $i . " ";
}
```

13.7. Boucle `foreach`

> La boucle `foreach` est spécifiquement conçue pour itérer sur les éléments d'un tableau (ou d'un objet itérable). Elle est souvent utilisée pour parcourir des collections de données.
> ***Exemple :***

```php
$couleurs = ["rouge", "vert", "bleu"];
foreach ($couleurs as $couleur) {
    echo $couleur." ";
}
```

13.8. Boucle `foreach` avec clé

> Vous pouvez également accéder à la clé (ou l'indice) de chaque élément lors de l'utilisation de la boucle `foreach`.
>
> ***Exemple :***

```php
$couleurs = ["rouge", "vert", "bleu"];
foreach ($couleurs as $couleur) {
    echo "Index: $key, Valeur: $color<br>";
}
```

14. Tableaux (Array)

14.1. Concept de tableau

> Un tableau, dans le contexte de la programmation, est une structure de données qui stocke une collection d'éléments. En PHP, comme dans de nombreux autres langages de programmation, les tableaux sont des outils essentiels et polyvalents pour gérer des ensembles de données.
>
> Un tableau peut stocker une liste d'éléments. Ces éléments peuvent être des nombres, des chaînes de caractères, des objets, voire d'autres tableaux.
>
> En PHP, les tableaux peuvent être utilisés pour diverses opérations telles que le tri de données, le regroupement d'informations, ou comme structure de données pour des algorithmes plus complexes.

14.2. Tableau Indexé

> Un **tableau indexé** est un type de tableau où chaque élément est automatiquement assigné à un index numérique, débutant généralement par 0.
>
> Cela signifie que le premier élément du tableau a un index de 0, le second un index de 1, et ainsi de suite.
>
> Les tableaux indexés sont idéaux pour stocker des séquences d'éléments où l'ordre est important et où l'accès se fait généralement par position.

14.3. Tableau Associatif

> Un **tableau associatif** est un type de tableau où chaque élément est associé à une clé unique.
>
> Contrairement aux tableaux indexés, les clés des tableaux associatifs peuvent être des chaînes de caractères ou des nombres, permettant une identification plus explicite et lisible des éléments.
>
> Les tableaux associatifs sont particulièrement utiles pour stocker et accéder à des données où chaque valeur est identifiée par une clé spécifique, comme un identifiant ou un nom.

14.4. Manipuler les tableaux

14.4.1. Création

```php
$tableauIndexe = array("pomme", "banane", "cerise");
$tableauIndexe = [1, 2, 3];
$tableauAssociatif = ["un" => 1, "deux" => 2, "trois" => 3];
```

14.4.2. Taille d'un tableau

```php
echo count($tableauIndexe);
echo count($tableauAssociatif);
```

14.4.3. Accès aux éléments

```php
echo $tableauIndexe[0];
echo $tableauAssociatif["deux"];
```

14.4.4. Modification des éléments

```php
$tableauIndexe[0] = 10;
$tableauAssociatif["deux"] = 20;
```

14.4.5. Ajout d'éléments

```php
$tableauIndexe[] = 4;
$tableauAssociatif["quatre"] = 4;
```

14.4.6. Suppression d'élément

> La fonction `array_splice()` en PHP permet de supprimer des éléments d'un tableau et, éventuellement, d'insérer de nouveaux éléments à leur place.
>
> Elle est plus flexible que `unset()` car elle peut supprimer plusieurs éléments à la fois et gérer des insertions et permet de supprimer des éléments d'un tableau tout en **conservant la structure** du tableau.
>
> La fonction `array_splice()` prend trois arguments principaux : le tableau à modifier, l'indice de départ à partir duquel vous souhaitez commencer la suppression et le nombre d'éléments à supprimer. Les éléments supprimés sont renvoyés dans un nouveau tableau.
> ***Exemple :***

```php
$tableau = [1, 2, 3, 4, 5];
$elementsSupprimes = array_splice($tableau, 2, 2);
```

14.4.7. Parcours des éléments d'un tableau avec la boucle `foreach()`

Parcours d'un tableau indexé :

> Nous utilisons `as` pour assigner chaque élément du tableau indexé `$tableauIndexe` à la variable `$element`, puis nous affichons chaque élément.

```php
$tableauIndexe = [1, 2, 3, 4, 5];
foreach ($tableauIndexe as $element) {
    echo $element . " ";
}
```

Parcours d'un tableau associatif :

> Nous utilisons `as` pour parcourir le tableau associatif `$tableauAssociatif` en assignant la clé à la variable `$cle` et la valeur à la variable `$valeur`. Ensuite, nous affichons chaque paire clé-valeur.

```php
$tableauAssociatif = ["un" => 1, "deux" => 2, "trois" => 3];
foreach ($tableauAssociatif as $cle => $valeur) {
    echo "Clé : " . $cle . ", Valeur : " . $valeur . "<br>";
}
```

14.5. Tableaux multidimensionnels

> Les tableaux multidimensionnels en PHP sont des tableaux qui contiennent d'autres tableaux en tant qu'éléments. Ils sont souvent utilisés pour représenter des données tabulaires ou des structures de données complexes.
>
> Les tableaux multidimensionnels sont utiles pour organiser et manipuler des données complexes, telles que les informations des étudiants dans une classe, les données de facturation, les résultats de sondages, etc.
>
> ***Exemple :***

```php
$classe = array(
    "eleve1" => array("nom" => "Jean", "age" => 18, "notes" => array(95, 88, 92)),
    "eleve2" => array("nom" => "Marie", "age" => 17, "notes" => array(89, 91, 78)),
    "eleve3" => array("nom" => "Pierre", "age" => 19, "notes" => array(75, 82, 96))
);
```

ou

```php
$classe = [
    "eleve1" => ["nom" => "Jean", "age" => 18, "notes" => [95, 88, 92]],
    "eleve2" => ["nom" => "Marie", "age" => 17, "notes" => [89, 91, 78]],
    "eleve3" => ["nom" => "Pierre", "age" => 19, "notes" => [75, 82, 96]]
];
```

Dans cet exemple, le tableau `$classe` est multidimensionnel, avec des éléments qui sont eux-mêmes des tableaux associatifs. Chaque élève est représenté par un tableau associatif avec des informations telles que le nom, l'âge et les notes, qui sont elles-mêmes stockées dans un tableau.
Vous pouvez accéder aux éléments d'un tableau multidimensionnel en utilisant des indices ou des

clés multiples. Par exemple, pour accéder à la note de Marie dans la deuxième matière (91), vous pouvez utiliser `$classe["eleve2"]["notes"][1]`.

14.5.1. Parcours d'un tableau multidimensionnel

> Pour parcourir un tableau multidimensionnel en PHP, vous pouvez utiliser des boucles `foreach` imbriquées pour accéder à chaque niveau du tableau.
> ***Exemple :***

```php
$classe = [
    "eleve1" => ["nom" => "Jean", "age" => 18, "notes" => [95, 88, 92]],
    "eleve2" => ["nom" => "Marie", "age" => 17, "notes" => [89, 91, 78]],
    "eleve3" => ["nom" => "Pierre", "age" => 19, "notes" => [75, 82, 96]]
];
foreach ($classe as $eleve => $informations) {
    echo "Élève : " . $eleve . "<br>";
    foreach ($informations as $cle => $valeur) {
        if (is_array($valeur)) {
            echo $cle . " : ";
            foreach ($valeur as $note) {
                echo $note . " ";
            }
            echo "<br>";
        } else {
            echo $cle . " : " . $valeur . "<br>";
        }
    }
    echo "<br>";
}
```

Sortie :

```
Élève : eleve1
nom : Jean
age : 18
notes : 95 88 92
Élève : eleve2
nom : Marie
age : 17
notes : 89 91 78
Élève : eleve3
nom : Pierre
age : 19
notes : 75 82 96
```

Dans cet exemple, nous utilisons une boucle `foreach` pour parcourir le tableau `$classe`. À chaque itération, nous obtenons les informations d'un élève sous forme d'un tableau associatif. Ensuite, nous utilisons une autre boucle `foreach` pour parcourir ces informations.
Si une valeur est un tableau (comme le tableau de notes), nous utilisons une boucle supplémentaire pour parcourir ce tableau. Sinon, nous l'affichons directement.

14.6. Fonctions utiles à utiliser avec les tableaux : `implode()` et `explode()`

14.6.1. `implode()`

> Convertit un tableau en une chaîne de caractères en utilisant un séparateur spécifié.
> ***Exemple :***

```php
$fruits = ["pomme", "banane", "orange"];
$chaine = implode(" - ", $fruits);
echo $chaine;
```

14.6.2. explode()

> Divise une chaîne de caractères en un tableau en utilisant un séparateur spécifié.
> ***Exemple :***

```php
$chaine = "pomme*banane*orange";
$fruits = explode("*", $chaine);
echo $fruits;
```

14.7. Tableaux et Fonctions Intégrées

Les fonctions de tableau intégrées en PHP sont un ensemble spécifique de fonctions intégrées fournies par le langage PHP pour faciliter la manipulation et la gestion des tableaux. Ces fonctions offrent diverses opérations qui peuvent être appliquées aux tableaux, allant de la manipulation basique des éléments à des opérations plus complexes.

Fonction	Description	Exemple d'utilisation
`count()`	Compte tous les éléments dans un tableau.	`count($tableau);`
`array_push()`	Ajoute un ou plusieurs éléments à la fin d'un tableau.	`array_push($tableau, "valeur1", "valeur2");`
`array_pop()`	Supprime le dernier élément d'un tableau.	`$element = array_pop($tableau);`
`array_shift()`	Supprime le premier élément d'un tableau.	`$premierElement = array_shift($tableau);`
`array_unshift()`	Ajoute un ou plusieurs éléments au début d'un tableau.	`array_unshift($tableau, "valeur1", "valeur2");`
`array_keys()`	Retourne toutes les clés ou un ensemble de clés d'un tableau.	`array_keys($tableau);`
`array_values()`	Retourne toutes les valeurs d'un tableau.	`array_values($tableau);`
`in_array()`	Vérifie si une valeur existe dans un tableau.	`in_array("valeur", $tableau);`
`array_search()`	Recherche une valeur dans un tableau et retourne sa clé.	`array_search("valeur", $tableau);`
`sort()`	Trie un tableau.	`sort($tableau);`
`rsort()`	Trie un tableau en ordre décroissant.	`rsort($tableau);`
`array_merge()`	Fusionne un ou plusieurs tableaux.	`array_merge($tableau1, $tableau2);`

Fonction	Description	Exemple d'utilisation
`array_slice()`	Extrait une portion de tableau.	`array_slice($tableau, 2, 3);`
`array_splice()`	Supprime une portion du tableau et la remplace.	`array_splice($tableau, 2, 1, "remplacement");`
`array_map()`	Applique une fonction à tous les éléments d'un tableau.	`array_map('fonction', $tableau);`
`array_filter()`	Filtre les éléments d'un tableau en utilisant une fonction.	`array_filter($tableau, 'fonction');`
`array_reduce()`	Réduit un tableau à une seule valeur en utilisant une fonction.	`array_reduce($tableau, 'fonction', $initial);`

14.8. Introduction aux Fonctions PHP

Les fonctions en PHP sont des blocs de code conçus pour effectuer une **tâche spécifique** et peuvent être **réutilisées** à travers un programme. Elles permettent de **structurer le code de manière modulaire**, rendant les applications **plus faciles à lire**, à **maintenir** et à **déboguer**.

Une fonction en PHP est déclarée avec le mot-clé `function`, suivi d'un nom unique pour la fonction, et peut accepter des **paramètres** qui influencent son exécution.

14.9. Déclaration de Fonction

La syntaxe de base pour la création et l'utilisation de fonctions en PHP est simple et directe, mais elle est fondamentale pour développer des applications efficaces et organisées.

Pour déclarer une fonction en PHP, utilisez le mot-clé `function`, suivi du nom de la fonction, d'une paire de parenthèses contenant zéro ou plusieurs paramètres, et enfin d'un bloc d'instructions entre accolades qui définissent le corps de la fonction.

Points clés à retenir :

- **Nom de la fonction** : Le nom doit être unique dans le contexte de son utilisation et il est recommandé d'utiliser un nom descriptif pour indiquer ce que la fonction fait.
- **Paramètres** : Les paramètres sont des variables listées dans les parenthèses de la déclaration de la fonction. Ils agissent comme des variables temporaires utilisées dans le corps de la fonction. Les paramètres sont optionnels; une fonction peut n'en avoir aucun.
- **Corps de la fonction** : C'est le bloc de code entre les accolades `{ }` qui est exécuté chaque fois que la fonction est appelée. Il peut contenir n'importe quel nombre d'instructions valides en PHP, y compris d'autres appels de fonctions.
  Exemple :

```
function nomDeLaFonction($parametre1, $parametre2) {

    echo "Ceci est une fonction avec des paramètres $parametre1 et $parametre2.";
}
```

14.10. Appel de Fonction

> Une fois qu'une fonction est définie, elle peut être exécutée (ou "appelée") en utilisant le nom de la fonction suivi de parenthèses.
>
> Si fonction attend des **paramètres** lors de sa déclaration, vous devez fournir des valeurs (**arguments**) pour chacun des paramètres attendus lors de son appel.

```
nomDeLaFonction("valeur1", "valeur2");
```

Exemple :

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

14.11. Paramètres de Fonction

> Les paramètres de fonction permettent de passer des valeurs à une fonction, afin qu'elle puisse effectuer des opérations en utilisant ces valeurs. PHP offre une grande flexibilité dans la manière de définir et d'utiliser les paramètres.

14.11.1. Paramètres par Défaut

> Il est possible de définir des valeurs par défaut pour les paramètres d'une fonction, ce qui permet de les rendre optionnels lors de l'appel de la fonction.
> ***Exemple :***

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

14.11.2. Passage par Valeur

> Par défaut, les paramètres en PHP sont passés par valeur, signifiant qu'une **copie de la variable** est passée à la fonction.
> ***Exemple :***

```php
function ajouterUn($nombre) {
    $nombre += 1;
}
$nombre = 10;
ajouterUn($nombre);
echo $nombre;
```

14.11.3. Passage par Référence

> Pour modifier la variable originale à l'**extérieur de la fonction**, les paramètres peuvent être passés par référence en utilisant le symbole `&`.
> ***Exemple :***

```php
function ajouterUn(&$nombre) {
    $nombre += 1;
}
$nombre = 10;
ajouterUn($nombre);
echo $nombre;
```

14.11.4. Typage des Paramètres

> Le typage des paramètres permet de spécifier le type attendu pour chaque paramètre, entraînant une erreur si le type ne correspond pas lors de l'appel de la fonction.
> ***Exemple :***

```php
function setAge(int $age) {
    echo "L'âge est $age.";
}
```

14.11.5. Type d'unions

> Depuis **PHP 8.0**, vous pouvez indiquer qu'un argument peut être de différents types en séparant chaque type par le caractère (pipe) `|` dans la déclaration de la fonction.
> ***Exemple :***

```php
function somme(int|float $a, int|float $b) {
    return $a + $b;
}
```

14.11.6. Paramètres Nommés

> Depuis **PHP 8.0**, les paramètres nommés permettent de passer des arguments à une fonction dans **n'importe quel ordre** en utilisant le nom des paramètres.
> ***Exemple :***

```php
function infoUtilisateur($prenom, $age) {
    echo "Prénom: $prenom, Age: $age";
}
infoUtilisateur(prenom: "Bob", age: 32 );
infoUtilisateur(age: 25, prenom: "Alice");
```

14.11.7. Fonctions Variadiques

> Les fonctions variadiques permettent de recevoir un **nombre variable d'arguments** grâce à l'opérateur `...`. Cet opérateur est appelé **opérateur de décomposition** ou opérateur de propagation ou encore `spread operator`.
> ***Exemple :***

```php
function somme(...$nombres) {
    return array_sum($nombres);
}
echo somme(1, 2, 3, 4);
```

14.12. Valeurs de Retour

> La valeur de retour d'une fonction est la valeur que **renvoie une fonction après son exécution**.

14.12.1. Types de Retour

> Les fonctions peuvent déclarer le type de leur valeur de retour, ce qui **renforce la sécurité du type** dans les applications.
> ***Exemple :***

```php
function solde(float $n1, float $n2, bool $remise, float $pourcentage = 10):
string
{

    $somme = $n1 + $n2;
```

```
    if ($remise) {

        $reduction = $somme * ($pourcentage / 100);
        $prixReduit = $somme - $reduction;
        return "La somme de votre achat est de $somme €, avec la réduction de $pourcentage%, vous devez payer maintenant la modique somme $prixReduit €.";
    } else {

        return "Prix à payer : $somme.";
    }
}
$n1 = 100;
$n2 = 50.5;
$remise = true;
$pourcentage = 15;
$message = solde($n1, $n2, $remise, $pourcentage);
echo $message;
```

14.13. Fonctions Anonymes et Closures

> Les fonctions anonymes, aussi connues sous le nom de `closures`, permettent la création de fonctions **sans nom spécifié**. Elles peuvent être utilisées comme **valeur de variable** et sont particulièrement utiles pour les **callbacks** et les fonctions de rappel.
> ***Exemple 1 :***

```
$saluer = function($nom) {
    echo "Bonjour, $nom!";
};
$saluer("Alice");
```

Exemple 2 :

```php
$addition = function($a, $b) {
    return $a + $b;
};
echo $addition(5, 3);
```

14.13.1. Fonctions Fléchées

Depuis **PHP 7.4**, les fonctions fléchées offrent une **syntaxe simplifiée** pour les fonctions anonymes, particulièrement quand elles contiennent une simple expression.

Elles permettent une **écriture plus concise** en **héritant automatiquement** les variables de leur **contexte parent**.
Exemple :

```php
$coefficient = 2;
$resultat = fn($valeur) => $valeur * $coefficient;
echo $resultat(5);
```

14.14. Portée des Variables

14.14.1. Variables Globales

Les variables globales :

- sont déclarées **à l'extérieur** de toutes les fonctions.
- peuvent être accessibles depuis n'importe quelle partie du script, y compris à l'intérieur des fonctions.
- pour accéder à une variable globale à l'intérieur d'une fonction, vous devez utiliser le mot-clé `global` avant de l'utiliser.
- il est aussi possible d'accéder à une variable globale via le tableau superglobal `$GLOBALS`.
  Exemple :

```php
$globalVar = "Je suis une variable globale";
function demoGlobal() {
    global $globalVar;
    echo $globalVar;
}
demoGlobal();
```

14.14.2. Variables locales

Les variables locales :

- sont définies **à l'intérieur d'une fonction** et ne peuvent être accessibles qu'à l'intérieur de cette même fonction.
- sont créées au moment où la fonction est appelée et **sont détruites lorsque la fonction a terminé son exécution**.
- L'utilisation de variables locales est une bonne pratique car elle aide à **éviter les conflits de noms de variables** dans des parties plus larges du programme et permet de **garder votre code propre et bien organisé.**
  Exemple :

```php
function demoFunction() {
    $localVar = "Je suis une variable locale";
    echo $localVar;
}
echo $localVar;
```

14.14.3. Variables Statiques

Les variables statiques c**onservent leur valeur entre les appels** de fonction.

Elles sont déclarées **à l'intérieur d'une fonction** mais **ne perdent pas leur valeur** lorsque l'exécution de la fonction est terminée.

La caractéristique principale d'une variable statique est **sa persistance** : sa valeur est mémorisée **entre les appels successifs** de la fonction, contrairement aux variables locales qui sont réinitialisées.
Exemple :

```php
function compteur() {
    static $compte = 0;
    $compte++;
    echo $compte . "\n";
}
compteur();
compteur();
compteur();
```