

PHP NATIF

Une introduction au monde backend

Mots clé : PHP, MySQL, Serveur, Apache, variables, tableaux, fonctions, boucles, sessions, cookies ...

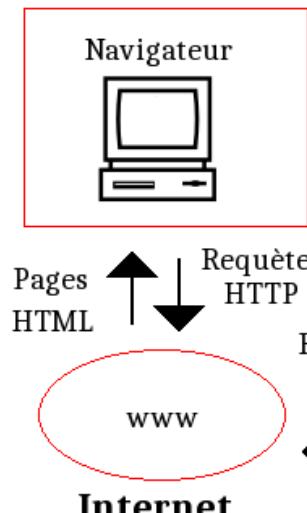
groupe DWWM

- M.TOULOUSE -

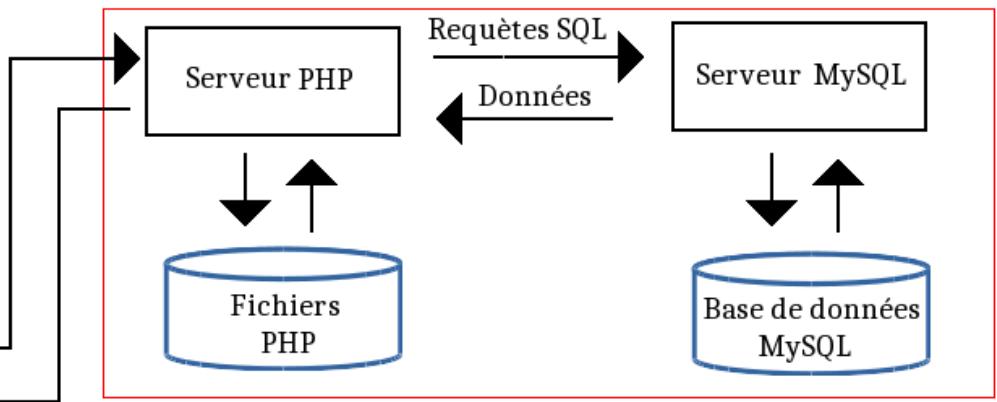
PHP NATIF

1. Environnement d'exécution & Outils

Internaute



Hébergeur de site Web



- LAMP, MAMP, XAMPP, WAMP, LARAGON sont des logiciels qui permettent de créer un serveur web local sur votre ordinateur.

2. Documentation

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

3. Visual Studio Code (extensions)

- Code Runner
- PHP DevSense
- PHP IntelliSense

4. 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`,

`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  
// Code PHP ici  
echo "Bonjour!";  
?>
```

php

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  
// Code PHP ici  
echo "Bonjour!";
```

php

5. echo

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

Exemple :

```
# GUILLEMETS
echo "Bonjour"; // guillemets simples
echo 'Bonjour'; // guillemets doubles

# AVEC PARENTHESES
echo ("Bonjour"); // guillemets simples
echo ('Bonjour'); // guillemets doubles

# échapper les guillemets doubles lorqu'ils se trouvent à l'intérieur d'une chaîne entre
guillemets doubles.
echo "Le language \"PHP\" est très agréable à apprêhender.';

# échapper les guillemets simples lorqu'ils se trouvent à l'intérieur d'une chaîne entre
guillemets simples.
echo 'Aujourd\'hui il fait très beau.';

# Chaîne multiples
echo 'Il fait', ' très beau ', 'ce matin.', '<br>';

# Concaténation
echo "Il fait" . " très beau ce matin !";
```

5.1. nl2br()

La fonction `nl2br` en PHP est utilisée pour convertir les sauts de ligne (`\n` ou `\r\n`) en balises HTML `
` 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() :

```

echo nl2br("Bonjour\n");
echo nl2br("Bonsoir\n");

// Bonjour
// Bonsoir

$texte = "Ceci est une phrase avec\n un saut de ligne.';

// Utilisation de nl2br pour convertir les sauts de ligne en balises <br>
$texte_formate = nl2br($texte);

// Affichage du texte formaté
echo $texte_formate;

// Ceci est une phrase avec
// un saut de ligne.

```

php

5.2. 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 `
` est appropriée.

Exemple :

```

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

```

php

5.3. 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;

```

php

6. Variables

6.1. 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).

6.2. Déclaration

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

php

6.3. Concaténation

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

php

6.4. Affichage

```
echo $age; // 25
echo $nom; // John
echo $est_majeur; // 1
echo $taille; // 1.75
```

php

Concaténation avec guillemets doubles + balise HTML :

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

php

Concaténation avec guillemets simples + balise HTML :

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

php

Guillemets doubles + balise HTML :

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

php

Guillemets doubles + balise HTML + {}

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

php

6.5. Destruction

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

php

6.6. 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 ?
<u>\$_GLOBALS</u>	Référence toutes les variables disponibles dans un contexte global	OUI
<u>\$_SERVER</u>	Variables de serveur et d'exécution	OUI
<u>\$_GET</u>	Variables HTTP GET	OUI
<u>\$_POST</u>	Variables HTTP POST	OUI
<u>\$_FILES</u>	Variable de téléchargement de fichier via HTTP	OUI
<u>\$_REQUEST</u>	Variables de requête HTTP	OUI
<u>\$_SESSION</u>	Variables de session	OUI
<u>\$_ENV</u>	Variables d'environnement	OUI
<u>\$_COOKIE</u>	Cookies HTTP	OUI
<u>\$_http_response_header</u>	En-têtes de réponse HTTP	NON
<u>\$_argc</u>	Le nombre d'arguments passés au script	NON
<u>\$_argv</u>	Tableau d'arguments passés au script	NON

6.7. 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
<code>\$_SERVER['DOCUMENT_ROOT']</code>	Racine du serveur
<code>\$_SERVER['HTTP_ACCEPT_LANGUAGE']</code>	Langage accepté par le navigateur
<code>\$_SERVER['HTTP_HOST']</code>	Nom de domaine du serveur
<code>\$_SERVER['HTTP_USER_AGENT']</code>	Type de navigateur
<code>\$_SERVER['PATH_INFO']</code>	Chemin WEB du script
<code>\$_SERVER['PATH_TRANSLATED']</code>	Chemin complet du script
<code>\$_SERVER['REQUEST_URI']</code>	Chemin du script
<code>\$_SERVER['REMOTE_ADDR']</code>	Adresse IP du client
<code>\$_SERVER['REMOTE_PORT']</code>	Port de la requête HTTP
<code>\$_SERVER['QUERY_STRING']</code>	Liste des paramètres passés au script
<code>\$_SERVER['SERVER_ADDR']</code>	Adresse IP du serveur
<code>\$_SERVER['SERVER_ADMIN']</code>	Adresse de l'administrateur du serveur
<code>\$_SERVER['SERVER_NAME']</code>	Nom local du serveur
<code>\$_SERVER['SERVER_SIGNATURE']</code>	Type de serveur
<code>\$_SERVER['REQUEST_METHOD']</code>	Méthode d'appel du script

7. 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.

Les constantes sont utiles pour stocker des informations qui ne doivent pas être modifiées durant l'exécution du script.

Les constantes sont globales. Elles sont accessibles partout dans le script, y compris dans les fonctions.

Les constantes ne peuvent pas être redéfinies. Une fois qu'une constante est définie, elle ne peut plus être modifiée ou supprimée.

Les constantes sont automatiquement converties en chaînes de caractères lorsqu'elles sont affichées avec `echo` ou `print`.

Les constantes sont automatiquement converties en nombres lorsqu'elles sont utilisées dans une opération mathématique.

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

php

7.1. Constantes pré-définies

```
echo PHP_VERSION;// Version de PHP
echo PHP_OS; // Système d'exploitation du serveur
echo PHP_INT_MAX; // Valeur maximale d'un entier
echo PHP_INT_MIN; // Valeur minimale d'un entier
echo PHP_INT_SIZE; // Taille d'un entier
echo PHP_FLOAT_MAX; // Valeur maximale d'un nombre à virgule
echo PHP_FLOAT_MIN; // Valeur minimale d'un nombre à virgule
echo PHP_FLOAT_DIG; // Nombre de chiffres significatifs d'un nombre à virgule
echo PHP_FLOAT_EPSILON; // Plus petite valeur positive pouvant être ajoutée à 1.0 pour obtenir
une valeur supérieure à 1.0
echo PHP_FLOAT_MIN; // Valeur minimale d'un nombre à virgule
```

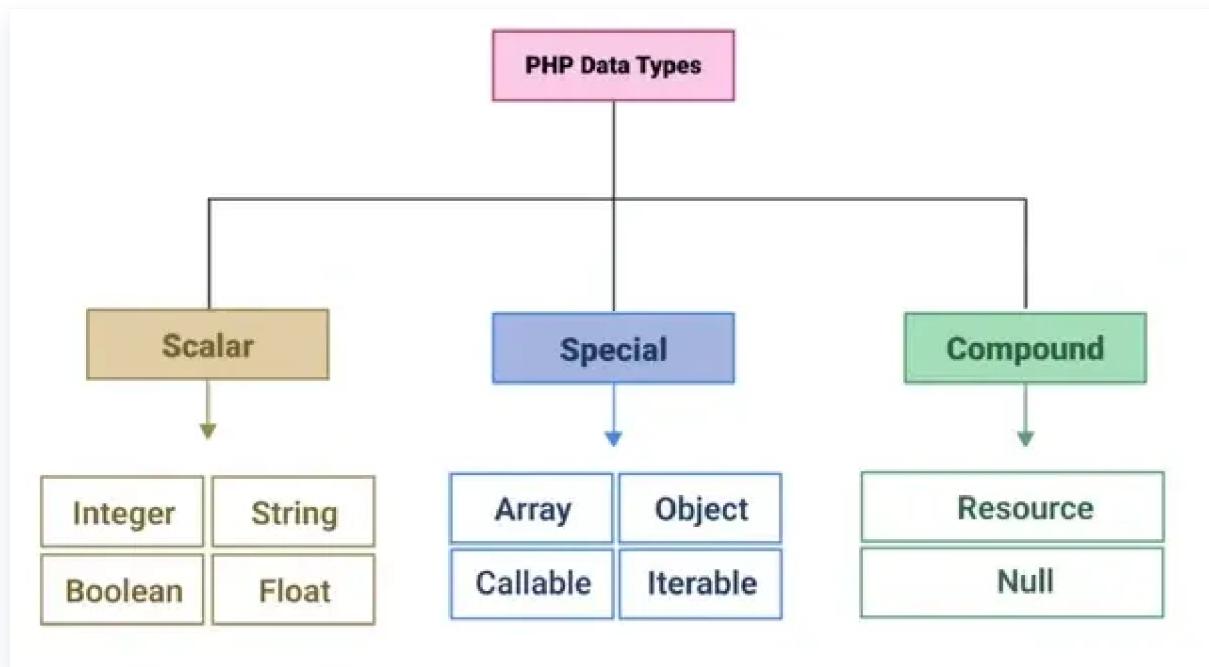
php

7.2. Constantes magiques pré-définies

```
echo __LINE__; // Numéro de la ligne
echo __FILE__; // Chemin absolu du fichier
echo __DIR__; // Chemin absolu du dossier
echo __FUNCTION__; // Nom de la fonction
echo __CLASS__; // Nom de la classe
echo __METHOD__; // Nom de la méthode
echo __NAMESPACE__; // Nom de l'espace de nom
echo __TRAIT__; // Nom du trait
echo __NAMESPACE__; // Nom de l'espace de nom
```

php

8. Types de données



8.1. Vue générale

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

8.2. 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.	<code>\$nombreEntier = 42;</code>
Float (nombre à virgule flottante)	Nombres décimaux avec une virgule flottante.	<code>\$prix = 19.99;</code>

Type de Données	Description	Exemple
Chaîne de Caractères (string)	Séquence de caractères, encadrée par des guillemets simples ou doubles.	<code>\$texte = "Bonjour, monde!";</code>
Booléen (bool)	Valeur de vérité, <code>true</code> (vrai) ou <code>false</code> (faux).	<code>\$estVrai = true;</code>
NULL	Variable sans valeur ou non initialisée.	<code>\$nulleVariable = null;</code>

8.3. 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).	<code>\$tableau = [1, 2, 3, 4];</code>
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.	-

8.4. 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.

```
$nombre = 42;
$type = gettype($nombre); // $type contiendra "integer"
```

php

- `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.

```
$nombre = "42";
settype($nombre, "integer"); // Change le type de $nombre en entier
```

php

9. Truthy et Falsy

En PHP, les valeurs qui sont considérées comme `true` (vrai) sont appelées `Truthy`, tandis que les valeurs qui sont considérées comme `false` (faux) sont appelées `Falsy`.

9.1. Valeurs `Truthy`

Type de Données	Exemples de Valeurs <code>Truthy</code>
Entier	42, -42, 0x42
Float	4.2, -4.2
Chaîne	"Hello", "0", "false"
Booléen	true
Tableau	[1, 2, 3]
Objet	new stdClass(), new DateTime()

9.2. Valeurs `Falsy`

Type de Données	Exemples de Valeurs <code>Falsy</code>
Entier	0, -0
Float	0.0, -0.0
Chaîne	"", "0", "false"
Booléen	false
Tableau	[]
Objet	null

10. Opérateurs

Les opérateurs en PHP sont des **symboles spéciaux** qui effectuent des opérations sur des variables et des valeurs. Ils sont utilisés pour effectuer des calculs mathématiques, des opérations logiques, des comparaisons, des affectations, etc.

10.1. Opérateurs Arithmétiques

Opérateur	Description	Exemple
+	Addition	<code>\$a + \$b</code>
-	Soustraction	<code>\$a - \$b</code>
*	Multiplication	<code>\$a * \$b</code>
/	Division	<code>\$a / \$b</code>

Opérateur	Description	Exemple
%	Modulo (reste)	\$a % \$b
**	Exponentiation	\$a ** \$b

- *Exemples :*

```
$a = 10;
$b = 5;

// Addition
$addition = $a + $b; // 15

// Soustraction
$soustraction = $a - $b; // 5

// Multiplication
$multiplication = $a * $b; // 50

// Division
$division = $a / $b; // 2

// Modulo
$modulo = $a % $b; // 0

// Exponentiation
$exponentiation = $a ** $b; // 100000
```

php

10.2. Opérateurs d'Assignment

Opérateur	Description	Exemple
=	Affectation	\$a = 10;
+=	Addition et affectation	\$a += 5;
-=	Soustraction et affectation	\$a -= 5;
*=	Multiplication et affectation	\$a *= 5;
/=	Division et affectation	\$a /= 5;
%=	Modulo et affectation	\$a %= 5;
**=	Exponentiation et affectation	\$a **= 5;

- *Exemples :*

```
$a = 10;

// Affectation
$b = $a; // $b = 10

// Addition et affectation
$a += 5; // $a = 15

// Soustraction et affectation
$a -= 5; // $a = 10

// Multiplication et affectation
$a *= 5; // $a = 50

// Division et affectation
$a /= 5; // $a = 10

// Modulo et affectation
$a %= 5; // $a = 0

// Exponentiation et affectation
$a **= 5; // $a = 0
```

php

10.3. Opérateurs d'incrémentation et de décrémentation

Opérateur	Description	Exemple
++	Incrémentation	\$a++;
--	Décrémentation	\$a--;
++	Pré-incrémentation	++\$a;
--	Pré-décrémentation	--\$a;

- *Exemples :*

```
// Nuance entre pré-incréméntation et post-incréméntation
$a = 5;
$b = $a++; // $a = 6, $b = 5
$b = ++$a; // $a = 7, $b = 7

// Nuance entre pré-décréméntation et post-décréméntation
$a = 5;
$b = $a--; // $a = 4, $b = 5
$b = --$a; // $a = 3, $b = 3
```

php

10.4. Opérateurs de Comparaison

Opérateur	Description	Exemple
==	Égalité	\$a == \$b

Opérateur	Description	Exemple
<code>==</code>	Identité	<code>\$a == \$b</code>
<code>!=</code>	Différence	<code>\$a != \$b</code>
<code><></code>	Différence	<code>\$a <> \$b</code>
<code>!==</code>	Non-identité	<code>\$a !== \$b</code>
<code><</code>	Inférieur	<code>\$a < \$b</code>
<code>></code>	Supérieur	<code>\$a > \$b</code>
<code><=</code>	Inférieur ou égal	<code>\$a <= \$b</code>
<code>>=</code>	Supérieur ou égal	<code>\$a >= \$b</code>
<code><=></code>	Spaceship	<code>\$a <=> \$b</code>

- *Exemples :*

```

$ a = 10;
$ b = 5;

// Égalité (vérifie la valeur)
$egalite = $a == $b; // false

// Identité (vérifie également le type)
$identite = $a === $b; // false

// Différence (vérifie la valeur)
$diff1 = $a != $b; // true

// Différence (vérifie la valeur)
$diff2 = $a <> $b; // true

// Non-identité (vérifie également le type)
$non_identite = $a !== $b; // true

// Inférieur (strictement)
$inferieur = $a < $b; // false

// Supérieur (strictement)
$superieur = $a > $b; // true

// Inférieur ou égal
$inf_egal = $a <= $b; // false

// Supérieur ou égal
$sup_egal = $a >= $b; // true

// Spaceship (nouveauté depuis PHP 7) : renvoie -1, 0 ou 1
// -1 si la valeur située à gauche de l'opérateur est inférieure à celle de droite
// 0 si les deux valeurs sont égales
// 1 si la valeur située à gauche de l'opérateur est supérieure à celle de droite
$spaceship1 = $a <=> $b; // 1 car $a est supérieur à $b
$spaceship2 = $b <=> $a; // -1 car $b est inférieur à $a
$spaceship3 = $a <=> $a; // 0 car $a est égal à $a

```

10.5. Opérateurs Logiques

Opérateur	Description	Exemple
and ou &&	ET logique	\$a && \$b
or ou	OU logique	\$a \$b
xor	OU exclusif	\$a xor \$b
!	NON logique	!\$a

- *Exemples :*

```
$a = true;
$b = false;

// ET logique
$et = $a && $b; // false

// OU logique
$ou = $a || $b; // true

// OU exclusif
$xor = $a xor $b; // true

// NON logique
$non = !$a; // false
```

php

10.6. Opérateurs de chaînes de caractères

Opérateur	Description	Exemple
.	Concaténation de chaînes	\$a . \$b
.=	Concaténation et affectation	\$a .= \$b

- *Exemples :*

```
$a = "Bonjour";
$b = "le monde !";

// Concaténation
$concat = $a . " " . $b; // "Bonjour le monde!"

// Concaténation et affectation
$a .= " " . "je m'appelle John"; // "Bonjour je m'appelle John"
```

php

10.7. Opérateur spéciaux

Opérateur	Description	Exemple
?	Opérateur ternaire classique	\$resultat = \$a > \$b ? \$a : \$b;
?:	Opérateur ternaire simplifié	\$resultat = \$a ?: \$b;
??	Opérateur de fusion null (null coalescing operator)	\$resultat = \$a ?? \$b;
??=	Opérateur d'affectation de fusion null	\$a ??= \$b;
=>	Opérateur flèche	\$tableau = ["nom" => "John", "age" => 25];
...	Opérateur de décomposition (spread operator)	[\$a, \$b] = [1, 2];

- *Exemples :*

```

$a = 10;
$b = 5;

// Opérateur ternaire classique
$resultat = $a > $b ? $a : $b; // 10

// Opérateur ternaire simplifié sur une exemple avec une enquête de satisfaction
$enquete_finie = false;
$reponse = $enquete_finie ?: "Enquête en cours..."; // "Enquête en cours..."

$nom = "";
$resultat = $nom ?: "Anonyme"; // Retourne "Anonyme" car $nom est "falsy"

// Opérateur de fusion null

# exemple 1 :
$resultat = $a ?? $b; // 10

# exemple 2 :
$prenom = null;
$etiquette = $nom ?? "Anonyme"; // "Anonyme"

// Opérateur d'affectation de fusion null

# exemple 1 :
$a ??= $b; // $a = 10

# exemple 2 :
$prenom = null;
$prenom ??= "Anonyme"; // $prenom = "Anonyme"

// Opérateur flèche
$tableau = ["nom" => "John", "age" => 25];

// Opérateur de décomposition (spread operator)
$array1 = [1, 2, 3];
$array2 = [4, 5];
$array3 = [6, 7, 8, 9, 10];

# Exemple 1 : Utilisation de l'opérateur `...` pour fusionner les tableaux
$result = [...$array1, ...$array2, ...$array3];
print_r($result); // Affiche : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Exemple 2 : Utilisation de l'opérateur `...` pour décomposer un tableau
[$a, $b, $c] = $array1;
echo $a; // 1
echo $b; // 2
echo $c; // 3

# Exemple 3 : Utilisation de l'opérateur `...` pour décomposer un tableau
[$a, $b, ...$c] = $array3;
echo $a; // 6
echo $b; // 7
print_r($c); // Affiche : [8, 9, 10]

```

```
# Exemple 4 : Utilisation de l'opérateur `...` pour décomposer un tableau
[$a, ...$b, $c] = $array3;
echo $a; // 6
print_r($b); // Affiche : [7, 8, 9]
echo $c; // 10

# Exemple 5 : appeler une fonction avec un nombre variable d'arguments
$arguments = [10, 20, 30];

// Appel de la fonction native `max` avec des arguments dynamiques
$maxValue = max(...$arguments);

echo $maxValue; // Affiche 30
```

11. Débogage

Le **débogage** est un processus essentiel dans le développement de logiciels qui consiste à **identifier et à corriger les erreurs** dans le code. En PHP, il existe plusieurs techniques et outils de débogage qui peuvent vous aider à résoudre les problèmes et à améliorer la qualité de votre code.

11.1. Affichage de messages

L'une des méthodes les plus simples pour déboguer en PHP est d'afficher des messages à l'écran pour suivre le flux d'exécution du programme et vérifier les valeurs des variables à différents points du code.

- **echo()** : La fonction `echo()` est utilisée pour afficher des messages ou des valeurs de variables à l'écran.

```
echo "Hello, World!";
```

- **print_r()** : La fonction `print_r()` est utilisée pour afficher des informations structurées sur une variable, y compris les tableaux et les objets.

```
$personne = ["nom" => "John", "age" => 25];
print_r($personne);
// Affiche : Array ( [nom] => John [age] => 25 )
```

php

- **var_dump()** : La fonction `var_dump()` est utilisée pour afficher des informations détaillées sur une ou plusieurs variables, y compris leur type et leur valeur.

```
$nombre = 42;
var_dump($nombre);
// Affiche : int(42)
```

php

11.2. Journalisation

La journalisation est une technique de débogage plus avancée qui consiste à enregistrer des messages, des erreurs et des informations sur l'exécution du programme dans un fichier ou une base de données. Cela permet de suivre l'historique des événements et de diagnostiquer les problèmes plus facilement.

- **error_log()** : La fonction `error_log()` est utilisée pour enregistrer des messages d'erreur, d'avertissement ou d'information dans le journal des erreurs de PHP.

```
$message = "Une erreur s'est produite.";
error_log($message);
```

php

12. Gestions des erreurs

La gestion des erreurs en PHP est un aspect important du développement de logiciels qui consiste à **gérer et à traiter les erreurs** qui se produisent pendant l'exécution du programme. Il existe plusieurs

12.1. Types d'erreurs

En PHP, il existe plusieurs types d'erreurs qui peuvent se produire pendant l'exécution d'un programme. Les erreurs les plus courantes sont les suivantes :

Type d'Erreur	Description
Erreur fatale	Erreur grave qui interrompt l'exécution du script.
Avertissement	Avertissement sur une condition potentiellement problématique.
Notice	Information sur une condition non critique.
Erreur	Erreur générée par l'utilisateur ou le programme.
Exception	Erreur générée par le programme à l'aide de blocs <code>try</code> , <code>catch</code> et <code>finally</code> (PHP 5 et supérieur).

- *Exemple :*

```
// Erreur fatale
echo $inconnu; // Affiche : PHP Notice: Undefined variable: inconnu in ...

// Avertissement
$nombre = "42";
$resultat = $nombre + 10; // Affiche : PHP Warning: A non-numeric value encountered in ...

// Notice
echo $hello; // Affiche : PHP Notice: Undefined variable: hello in ...

// Erreur
$division = 10 / 0; // Affiche : PHP Warning: Division by zero in ...

// Exception
try {
    throw new Exception("Une erreur s'est produite.");
} catch (Exception $e) {
    echo "Erreur : " . $e->getMessage(); // Affiche : Erreur : Une erreur s'est produite.
}
```

php

12.2. Gestion des erreurs

En PHP, il existe plusieurs techniques pour gérer les erreurs et les exceptions qui se produisent pendant l'exécution d'un programme. Les blocs `try`, `catch` et `finally` sont utilisés pour gérer les exceptions, tandis que les fonctions `error_reporting()`, `set_error_handler()` et `set_exception_handler()` sont utilisées pour gérer les erreurs.

- **`error_reporting()`** : La fonction `error_reporting()` est utilisée pour définir le niveau de rapport d'erreurs pour le script PHP. Elle prend en argument un entier représentant le niveau de rapport d'erreurs.

```
error_reporting(E_ALL); // Affiche toutes les erreurs
```

php

- **`set_error_handler()`** : La fonction `set_error_handler()` est utilisée pour définir une fonction de gestion des erreurs personnalisée qui sera appelée à chaque fois qu'une erreur se produit.

```

function gestion_erreur($code, $message, $fichier, $ligne) {
    echo "Une erreur est survenue :<br>";
    echo "Code : $code<br>";
    echo "Message : $message<br>";
    echo "Fichier : $fichier<br>";
    echo "Ligne : $ligne<br>";
}

set_error_handler("gestion_erreur"); // Appelle la fonction gestion_erreur() en cas d'erreur

// Test avec une erreur
$test = 10 / 0; // Génère une erreur de division par zéro et appelle la fonction
gestion_erreur() pour afficher finalement le message d'erreur personnalisé suivant :
// Une erreur est survenue :
// Code : 2
// Message : Division by zero
// Fichier : /var/www/html/test.php
// Ligne : 20

```

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

13.1. 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

// Conversion de Chaînes de Caractères en Nombres
$chaine = "42";
$nombre = $chaine + 10;
echo "Conversion de chaîne en nombre : $nombre\n"; // 52

// Conversion de Nombres en Chaînes de Caractères
$nombre = 42;
$chaine = "Le nombre est " . $nombre;
echo "Conversion de nombre en chaîne : $chaine\n"; // 42

// Conversion de Booléens en Nombres
$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";
// vrai_entier = 11, faux_entier = 10

// Conversion entre Types Numériques
$entier = 42;
$flottant = $entier / 2;
echo "Conversion entre types numériques : flottant = $flottant\n"; // 21.0

// Conversion de Chaînes de Caractères en Booléens
$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";
// Affiche : Conversion de chaînes en booléens : est_vrai = 1, est_faux =
// Note : est_faux est vide car une chaîne vide est évaluée comme false en booléen

?>
```

13.2. 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

// Conversion en Entier (int)
$nombre = 42.75;
$entier = (int)$nombre;
echo "Conversion en entier : $entier\n"; // Conversion en entier : 42

// Conversion en Nombre à Virgule Flottante (float)
$chaine = "3.14";
$flottant = (float)$chaine;
echo "Conversion en flottant : $flottant\n"; // Conversion en flottant : 3.14

// Conversion en Chaîne de Caractères (string)
$nombre = 42;
$chaine = strval($nombre);
echo "Conversion en chaîne : $chaine\n"; // Conversion en chaîne : 42

// Conversion en Booléen (bool)
$valeur = 0;
$boolean = (bool)$valeur;
echo "Conversion en booléen : " . ($boolean ? "true" : "false") . "\n"; // Conversion en
booléen : false

?>
```

13.3. Fonctions de conversion explicite de types

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

```
<?php

// Conversion en Entier (int)
$chaine = "42";
$entier = intval($chaine);
echo "Conversion en entier : $entier\n"; // Conversion en entier : 42

// Conversion en Nombre à Virgule Flottante (float)
$chaine = "3.14";
$flottant = floatval($chaine);
echo "Conversion en flottant : $flottant\n"; // Conversion en flottant : 3.14

// Conversion en Chaîne de Caractères (string)
$nombre = 42;
$chaine = strval($nombre);
echo "Conversion en chaîne : $chaine\n"; // Conversion en chaîne : 42

?>
```

14. 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.

14.1. 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";
}

// aucune sortie
```

php

14.2. 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 :

```
$note = 75;

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

// B
```

php

14.3. 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 :

```
$note = 75;

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

// C
```

php

14.4. 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 :

```
$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.";
}

// C'est le début de la semaine
```

php

14.5. 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.

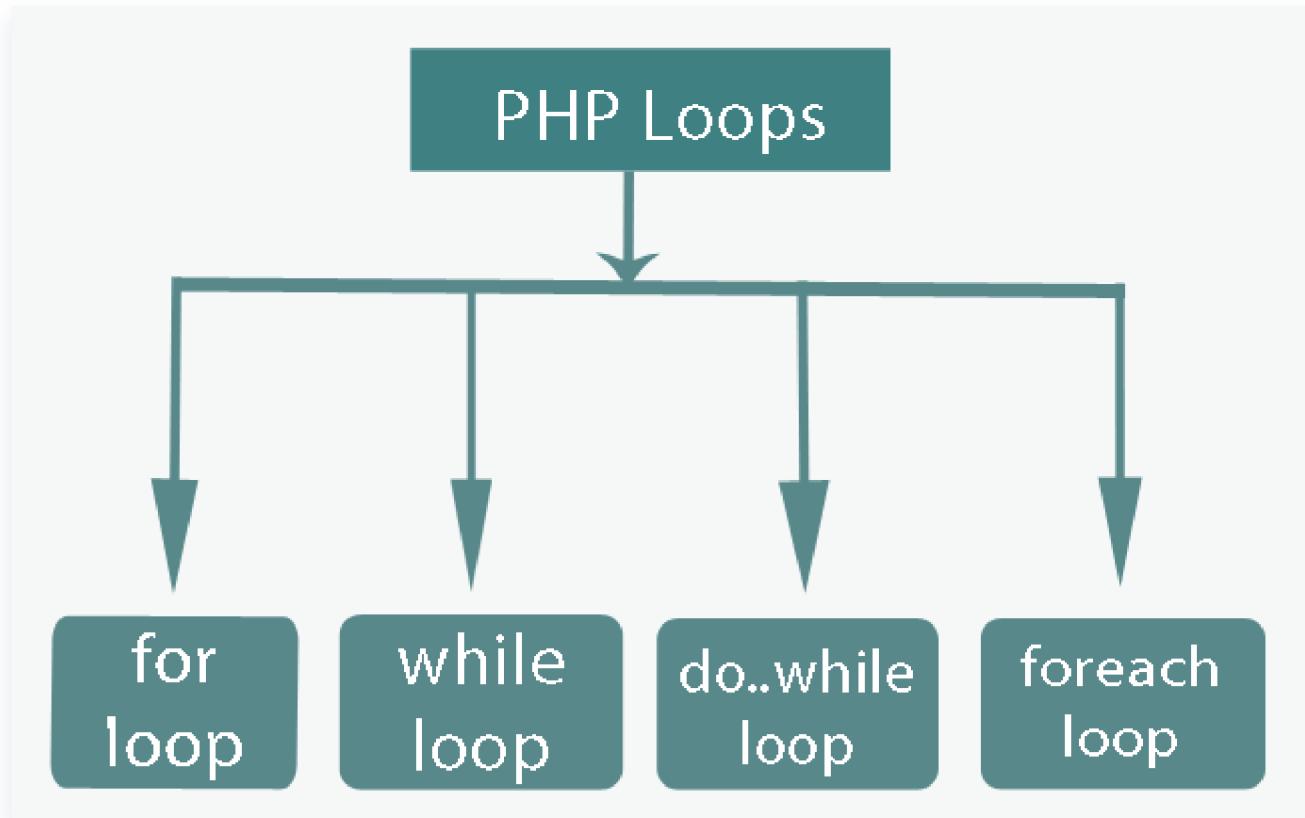
Exemple :

```
$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;  
  
// C'est le deuxième jour de la semaine.
```

php

15. Boucles

Les boucles contrôlent la manière dont le flux du programme est répété ou itéré. Elles permettent 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. **Incrémantation ou Décrémentation** : modifier la variable de contrôle à chaque itération pour éventuellement atteindre la condition d'arrêt.

15.1. Boucle `for`

La boucle `for` est idéale lorsque vous connaissez le nombre d'itérations à l'avance.

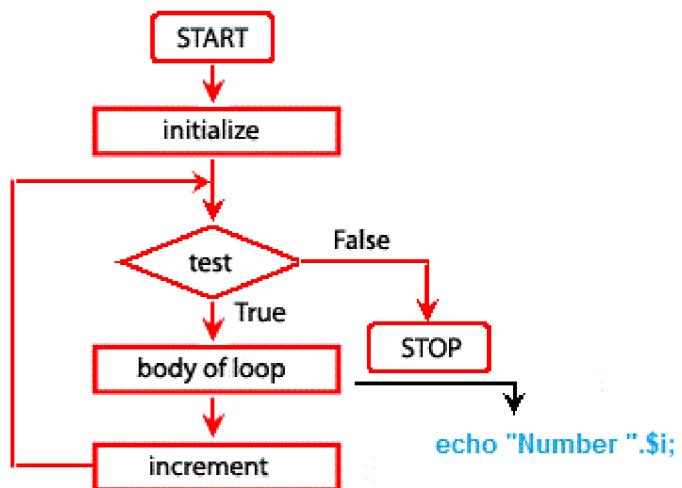
Elle se compose de trois parties : l'initialisation, la condition de continuation et l'incrémentation.

Example :

```

setup loop    test loop    increment loop
for ( $i = 0; $i < 5; $i++ ){
    echo $i;
}
    do some work
  
```

Flow Chart



Exemple :

```

for ($i = 0; $i < 5; $i++) {
    echo $i; // Code exécuté à chaque itération
}
// 1234
  
```

php

15.2. 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 :

```

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

// Boucle externe : 1, Boucle interne : 1
// Boucle externe : 1, Boucle interne : 2
// Boucle externe : 1, Boucle interne : 3
// Boucle externe : 2, Boucle interne : 1
// Boucle externe : 2, Boucle interne : 2
// Boucle externe : 2, Boucle interne : 3
// Boucle externe : 3, Boucle interne : 1
// Boucle externe : 3, Boucle interne : 2
// Boucle externe : 3, Boucle interne : 3

```

php

Exemple :

```

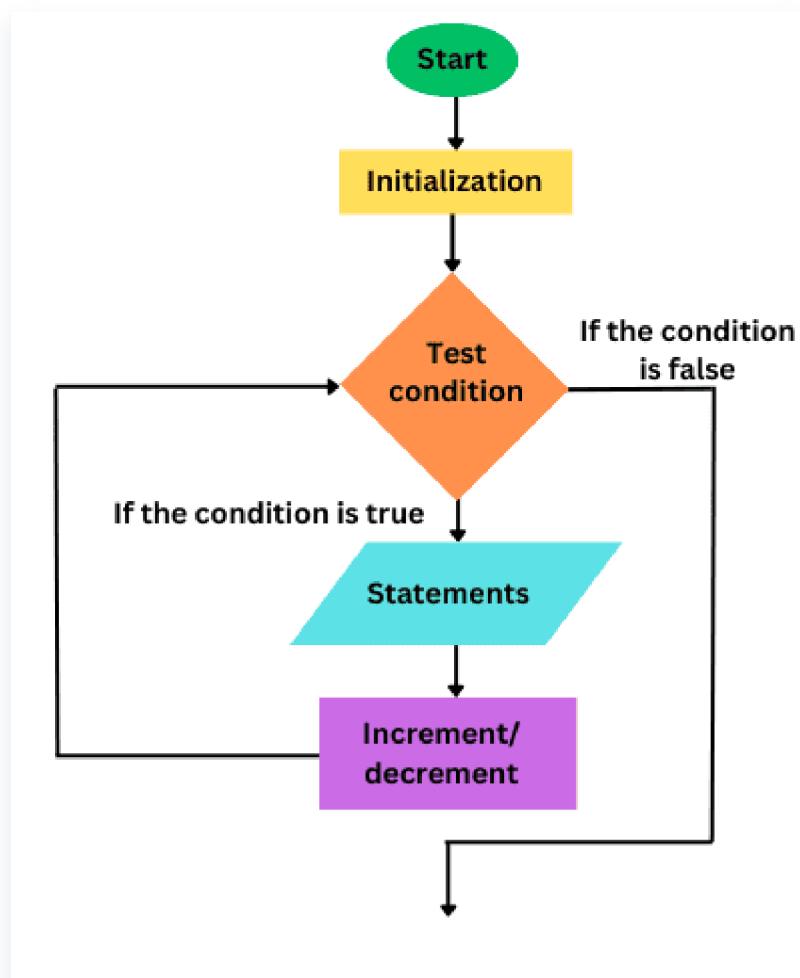
for ($i = 1; $i <= 10; $i++) {
    for ($j = 1; $j <= 10; $j++) {
        echo "$i x $j = " . ($i * $j) . " ";
    }
    echo "\n"; // Saut de ligne entre chaque table de multiplication
}

```

php

15.3. 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.



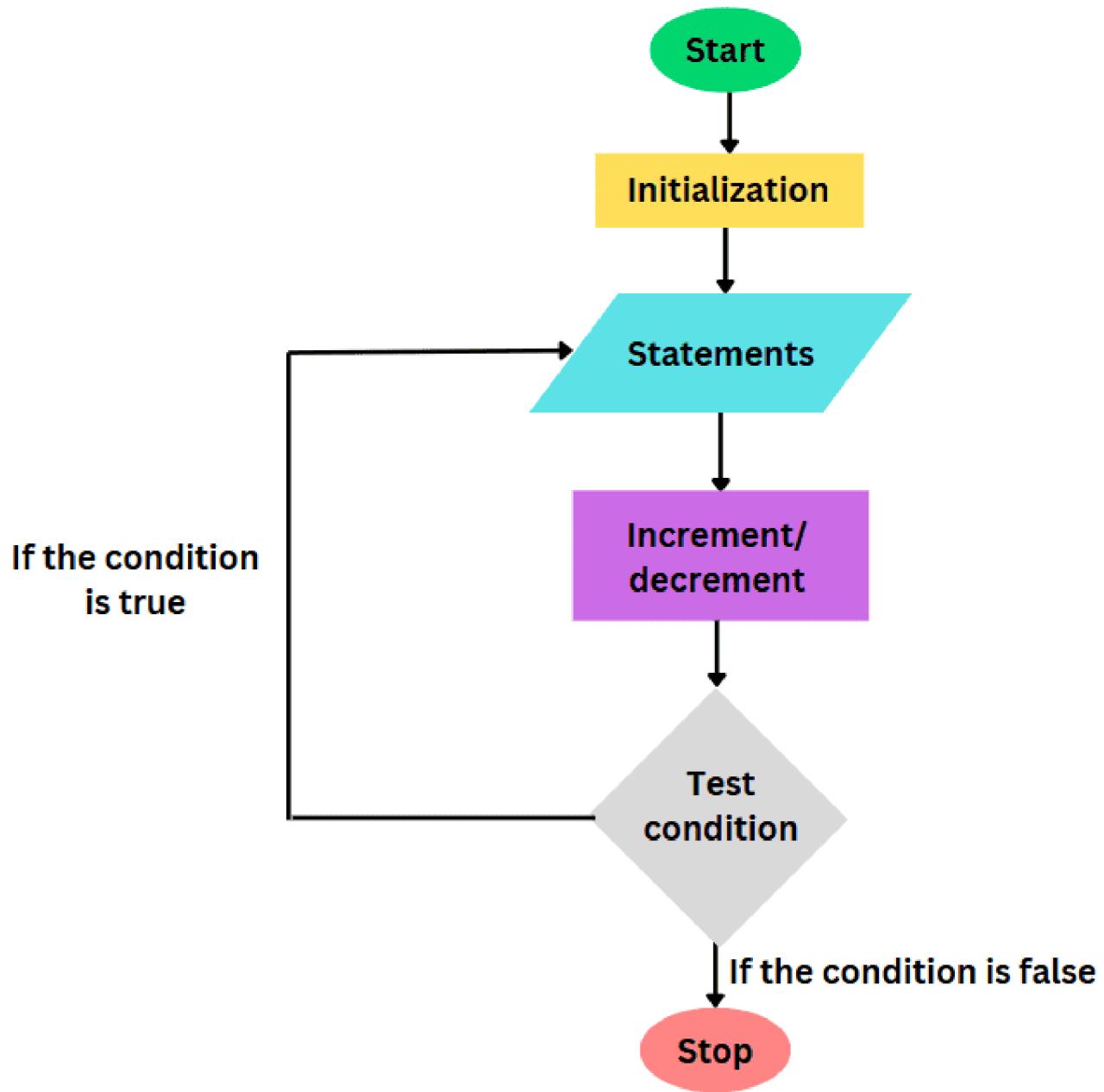
Exemple :

```
$i = 9;
while ($i > 5) {
    echo $i; // Code exécuté à chaque itération
    $i--;
}
// 9876
```

php

15.4. 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.



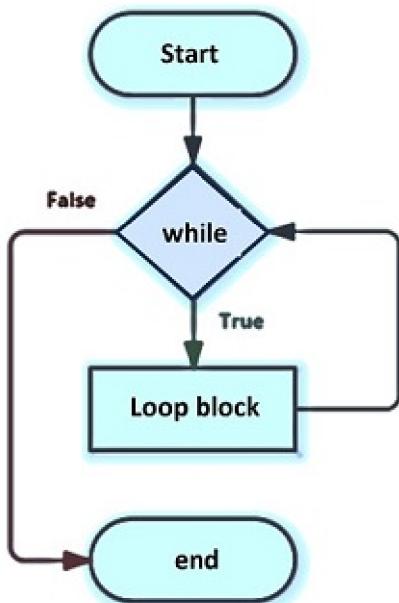
Exemple :

```
$i = 9;
do {
    echo $i; // Code exécuté à chaque itération
    $i--;
} while ($i < 5);
// 9876
```

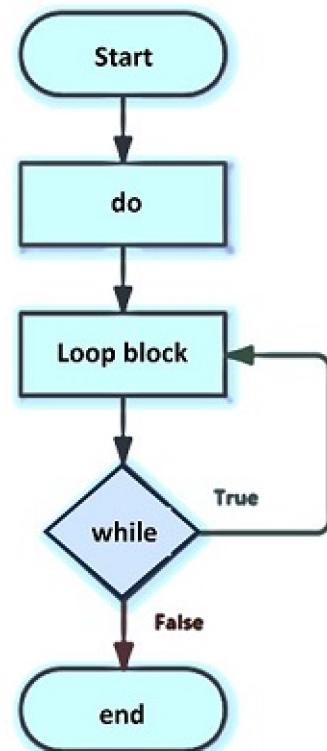
php

15.5. Comparaison while et do .. while

WHILE



DO-WHILE



15.6. 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`.

```

for ($i = 1; $i <= 5; $i++) {
    if ($i == 3) {
        break; // Sort de la boucle lorsque $i atteint 3
    }
    echo $i . " ";
}

// 1 2
  
```

php

15.7. 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`.

```
for ($i = 1; $i <= 5; $i++) {
    if ($i == 3) {
        continue; // Passe à l'itération suivante lorsque $i est égal à 3
    }
    echo $i . " ";
}

// 1 2 4 5
```

php

15.8. 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 :

```
$couleurs = ["rouge", "vert", "bleu"];
foreach ($couleurs as $couleur) {
    echo $couleur." "; // Code exécuté à chaque itération
}
// rouge vert bleu
```

php

15.9. 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 :

```
$couleurs = ["rouge", "vert", "bleu"];
foreach ($couleurs as $couleur) {
    echo "Index: $key, Valeur: $color<br>"; // Code à exécuté pour chaque élément avec sa clé
}
// Index: 0, Valeur: rouge
// Index: 1, Valeur: vert
// Index: 2, Valeur: bleu
```

php

16. Les Tableaux (Array)

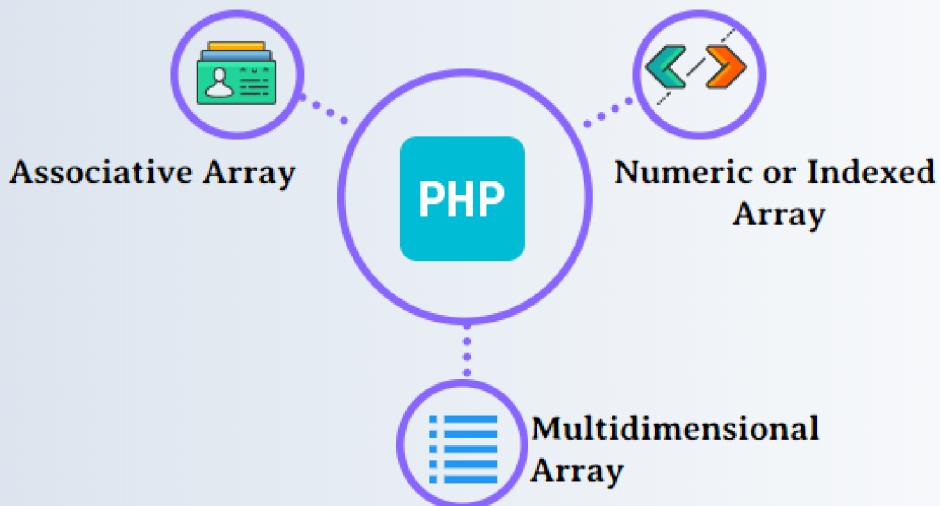
16.1. Concept de tableau

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

Arrays in PHP

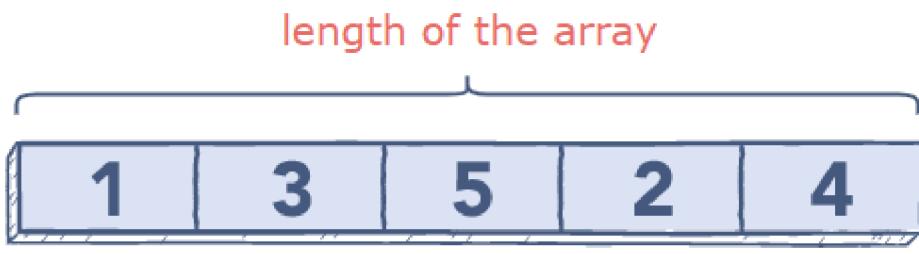


16.2. Tableau Indexé

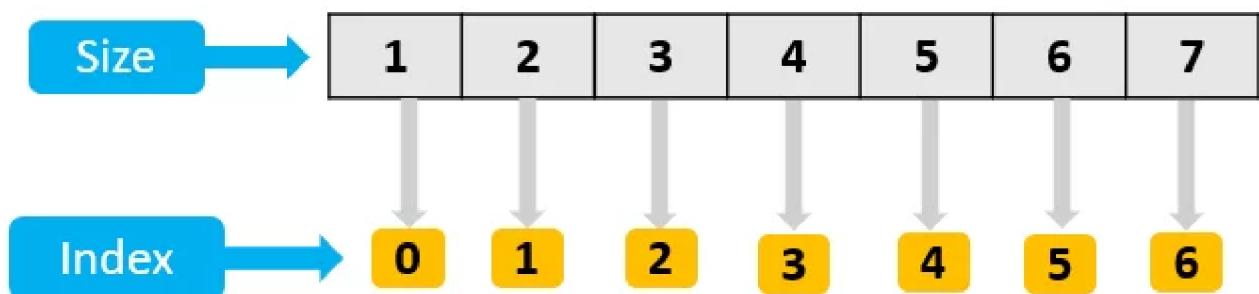
Un **tableau indexé** est un type de tableau où chaque élément est automatiquement assigné à un index numérique, débutant généralement par 0.

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.



PHP sizeof Function



`$mytestArray`

`sizeof($mytestArray)`

7

16.3. 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.

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.

16.4. Manipuler les tableaux

16.4.1. Création

```
$tableauIndexe = array("pomme", "banane", "cerise"); // Tableau indexé en syntaxe classique
$tableauIndexe = [1, 2, 3]; // Tableau indexé en syntaxe raccourcie
$tableauAssociatif = ["un" => 1, "deux" => 2, "trois" => 3]; // Tableau associatif
```

php

16.4.2. Taille d'un tableau

```
echo count($tableauIndexe); // Affiche le nombre d'éléments du tableau indexé : 3
echo count($tableauAssociatif); // Affiche le nombre de paires clé-valeur du tableau associatif
: 3
```

php

16.4.3. Accès aux éléments

```
echo $tableauIndexe[0]; // Accède au premier élément du tableau indexé (affiche 1)
echo $tableauAssociatif["deux"]; // Accède à la valeur associée à la clé "deux" (affiche 2)
```

php

16.4.4. Modification des éléments

```
$tableauIndexe[0] = 10; // Modifie le premier élément du tableau indexé
$tableauAssociatif["deux"] = 20; // Modifie la valeur associée à la clé "deux"
```

php

16.4.5. Ajout d'éléments

```
$tableauIndexe[] = 4; // Ajoute 4 à la fin du tableau indexé
$tableauAssociatif["quatre"] = 4; // Ajoute une paire clé-valeur au tableau associatif
```

php

16.4.6. 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 :

```
// Tableau de départ
$tableau = [1, 2, 3, 4, 5];

// Supprimer deux éléments à partir de l'indice 2
$elementsSupprimes = array_splice($tableau, 2, 2);

// $tableau maintenant : [1, 2, 5]
// $elementsSupprimes : [3, 4]
```

php

16.4.7. 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.

```
$tableauIndexe = [1, 2, 3, 4, 5];

foreach ($tableauIndexe as $element) {
    echo $element . " "; // Affiche chaque élément du tableau indexé
}
```

php

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.

```
$tableauAssociatif = ["un" => 1, "deux" => 2, "trois" => 3];

foreach ($tableauAssociatif as $cle => $valeur) {
    echo "Clé : " . $cle . ", Valeur : " . $valeur . "<br>"; // Affiche chaque paire clé-valeur
} 
```

php

16.5. 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 :

```
$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))
); 
```

php

ou

```
$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]]
]; 
```

php

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]`.

16.5.1. 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 :

```

$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>";
}

```

php

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

```

shell

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.

16.6. Fonctions utiles à utiliser avec les tableaux : *implode()* et *explode()*

16.6.1. *implode()*

Convertit un tableau en une chaîne de caractères en utilisant un séparateur spécifié.

Exemple :

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

php

16.6.2. *explode()*

Divise une chaîne de caractères en un tableau en utilisant un séparateur spécifié.

Exemple :

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

php

16.7. 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
<code>count()</code>	Compte tous les éléments dans un tableau.	<code>count(\$tableau);</code>
<code>array_push()</code>	Ajoute un ou plusieurs éléments à la fin d'un tableau.	<code>array_push(\$tableau, "valeur1", "valeur2");</code>
<code>array_pop()</code>	Supprime le dernier élément d'un tableau.	<code>\$element = array_pop(\$tableau);</code>
<code>array_shift()</code>	Supprime le premier élément d'un tableau.	<code>\$premierElement = array_shift(\$tableau);</code>
<code>array_unshift()</code>	Ajoute un ou plusieurs éléments au début d'un tableau.	<code>array_unshift(\$tableau, "valeur1", "valeur2");</code>
<code>array_keys()</code>	Retourne toutes les clés ou un ensemble de clés d'un tableau.	<code>array_keys(\$tableau);</code>
<code>array_values()</code>	Retourne toutes les valeurs d'un tableau.	<code>array_values(\$tableau);</code>
<code>in_array()</code>	Vérifie si une valeur existe dans un tableau.	<code>in_array("valeur", \$tableau);</code>
<code>array_search()</code>	Recherche une valeur dans un tableau et retourne sa clé.	<code>array_search("valeur", \$tableau);</code>
<code>sort()</code>	Trie un tableau.	<code>sort(\$tableau);</code>

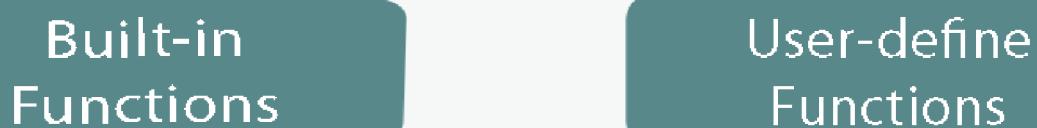
Fonction	Description	Exemple d'utilisation
<code>rsort()</code>	Trie un tableau en ordre décroissant.	<code>rsort(\$tableau);</code>
<code>array_merge()</code>	Fusionne un ou plusieurs tableaux.	<code>array_merge(\$tableau1, \$tableau2);</code>
<code>array_slice()</code>	Extrait une portion de tableau.	<code>array_slice(\$tableau, 2, 3);</code>
<code>array_splice()</code>	Supprime une portion du tableau et la remplace.	<code>array_splice(\$tableau, 2, 1, "remplacement");</code>
<code>array_map()</code>	Applique une fonction à tous les éléments d'un tableau.	<code>array_map('fonction', \$tableau);</code>
<code>array_filter()</code>	Filtre les éléments d'un tableau en utilisant une fonction.	<code>array_filter(\$tableau, 'fonction');</code>
<code>array_reduce()</code>	Réduit un tableau à une seule valeur en utilisant une fonction.	<code>array_reduce(\$tableau, 'fonction', \$initial);</code>

17. Fonctions

17.1. 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.



17.2. 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.

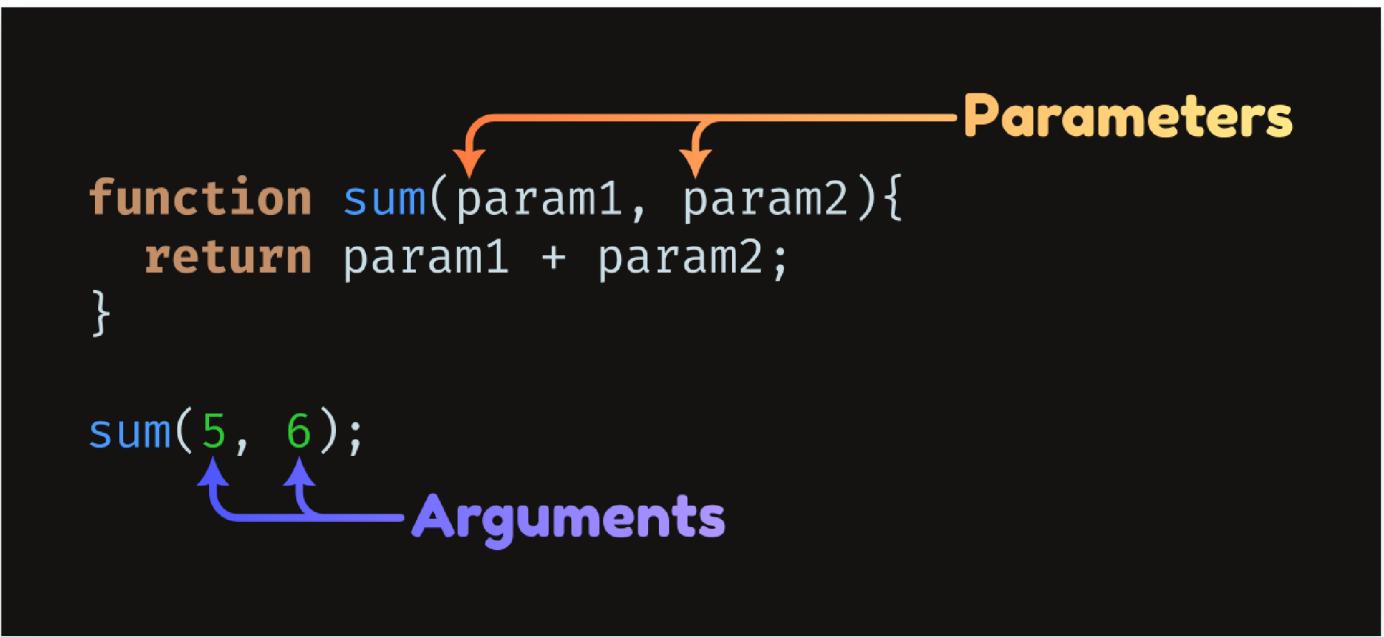
Pour déclarer une fonction en PHP, utilisez le mot-clé `function`, suivi du nom de la fonction, d'une paire de parenthèses contenant zéro ou plusieurs paramètres, et enfin d'un bloc d'instructions entre accolades qui définissent le corps de la fonction.

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) {
    // Corps de la fonction
    echo "Ceci est une fonction avec des paramètres $parametre1 et $parametre2.";
}
```



17.3. 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 la 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");
```

php

Exemple :

```
// Déclaration de la fonction
function saluer($nom) {
    echo "Bonjour, $nom!";
}

// Appel de la fonction
saluer("Alice");

// Affiche : Bonjour, Alice!
```

php

17.4. 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.

17.4.1. 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 :

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

saluer(); // Affiche : Bonjour, Visiteur!
saluer("Alice"); // Affiche : Bonjour, Alice!
```

php

17.4.2. 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 :

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

$nombre = 10;

ajouterUn($nombre);

echo $nombre; // Affiche : 10
```

php

17.4.3. 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 :

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

$nombre = 10;

ajouterUn($nombre);

echo $nombre; // Affiche : 11
```

php

17.4.4. 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 :

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

php

17.4.5. 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 :

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

php

17.4.6. 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 :

```
function infoUtilisateur($prenom, $age) {
    echo "Prénom: $prenom, Age: $age";
}

infoUtilisateur(prenom: "Bob", age: 32); // Prenom: Bob, Age : 32
infoUtilisateur(age: 25, prenom: "Alice"); // Prénom: Alice, Age : 25
```

php

17.4.7. 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 :

```
function somme(...$nombres) {
    return array_sum($nombres);
}

echo somme(1, 2, 3, 4); // Affiche : 10
```

php

17.5. Valeurs de Retour

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

17.5.1. 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 :

```

function solde(float $n1, float $n2, bool $remise, float $pourcentage = 10): string
{
    // Calculez la somme des deux nombres
    $somme = $n1 + $n2;

    // Vérifiez si la réduction doit être appliquée (booléen)
    if ($remise) {
        // Calculez la réduction en pourcentage
        $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 {
        // Si la réduction n'est pas appliquée, retournez simplement la somme
        return "Prix à payer : $somme.";
    }
}

// Utilisation de la fonction
$n1 = 100;
$n2 = 50.5;
$remise = true;
$pourcentage = 15;

$message = solde($n1, $n2, $remise, $pourcentage);

echo $message;

```

17.6. 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"); // Affiche : Bonjour, Alice!

```

Exemple 2:

```
$addition = function($a, $b) {
    return $a + $b;
};

echo $addition(5, 3); // Affiche : 8
```

php

17.6.1. 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 :

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

echo $resultat(5); // Affiche : 10
```

php

17.7. Portée des Variables

17.7.1. 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 :

```
$globalVar = "Je suis une variable globale";

function demoGlobal() {
    global $globalVar; // Déclare $globalVar comme étant globale à l'intérieur de cette
fonction
    echo $globalVar; // Accède à la variable globale
}

demoGlobal(); // Je suis une variable globale
```

php

17.7.2. 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 :

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

echo $localVar; // Erreur : la variable locale est inaccessible ici.
```

php

17.7.3. Variables Statiques

Les variables statiques **conservent 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 :

```
function compteur() {
    static $compte = 0; // Initialisée seulement lors du premier appel
    $compte++;
    echo $compte . "\n";
}

compteur(); // Affiche : 1
compteur(); // Affiche : 2
compteur(); // Affiche : 3
```

php

18. Formulaires

Les formulaires web sont des éléments d'interface utilisateur qui permettent de **collecter des informations** auprès des visiteurs d'un site web.

Ils sont essentiels pour des interactions telles que les **inscriptions**, les **connexions**, les **recherches** et les **soumissions** de commentaires.

18.1. Requêtes HTTP

Une requête HTTP (Hypertext Transfer Protocol request) est une demande faite par un client (comme un navigateur web ou une application) à un serveur web pour récupérer ou envoyer des données. Les requêtes HTTP sont utilisées pour interagir avec des ressources web, telles que des pages web, des images, des vidéos, des services web, etc.

Une requête HTTP se compose généralement des éléments suivants :

1. Méthode : La méthode HTTP spécifie l'action à effectuer sur la ressource. Les méthodes courantes incluent GET (pour récupérer des données), POST (pour envoyer des données), PUT (pour mettre à jour des données), DELETE (pour supprimer des données), etc.
2. URL (Uniform Resource Locator) : L'URL identifie la ressource ciblée par la requête. Il contient l'adresse du serveur et le chemin vers la ressource spécifique sur ce serveur.
3. En-têtes (headers) : Les en-têtes HTTP contiennent des informations supplémentaires sur la requête, telles que les informations sur le navigateur, les cookies, les informations de sécurité, etc.
4. Corps de la requête : Dans le cas des méthodes POST et PUT, le corps de la requête peut contenir des données à envoyer au serveur, par exemple, des données de formulaire.
5. Version HTTP : La version du protocole HTTP utilisée, par exemple, HTTP/1.1.

Une fois que le serveur reçoit la requête, il la traite en fonction de la méthode et de l'URL, puis renvoie une réponse au client. La réponse contient également des en-têtes et un corps de réponse qui peut contenir les données demandées ou des informations sur l'état de la requête.

Les requêtes HTTP sont au cœur de la communication sur le web, permettant aux clients de demander et de recevoir des données depuis des serveurs web, ce qui rend possible la navigation sur Internet et l'interaction avec des services en ligne.

18.2. Fonctionnement des formulaires web

18.2.1. Formulaires HTML

Un formulaire est défini en HTML par la balise `<form>` et peut contenir divers types d'entrées (input), telles que du texte, des mots de passe, des boutons radio, des cases à cocher, etc.

18.2.2. Méthodes GET et POST

Lorsqu'un client communique avec un serveur sur le Web, ce processus est activé par le protocole de transfert hypertexte HTTP (protocole de réponse à une demande entre un client et un serveur).

Les méthodes GET et POST sont les deux plus courantes méthodes de demande HTTP.

Elles sont utilisées pour récupérer ou envoyer des données à un serveur.

Elles font partie intégrante du modèle client-serveur qui permet la communication entre un client et un serveur via le World Wide Web (WWW).

19. FIN

Fin du cours !