

M 107 : Développer des sites web dynamiques

1 ère année

Filière : Développement Digital (Tronc commun)



.....

Partie 2: Programmer en PHP



.....

Chapitre 1: Maitriser le langage PHP



PHP Intégration d'un script dans une page

- Les pages web sont au format html. Les pages web dynamiques générées avec PHP sont au format php. Le code source php est directement insérer dans le fichier html grâce au conteneur de la norme XML : `<?php ... ?>`
- **Exemple**

```
<html>
  <body>
    <?php
      echo "Bonjour";
    ?>
  </body>
</html>
```

Tout ce qui est à l'extérieur de la balise PHP est transmis tel qu'il est au navigateur; seul les instructions PHP sont interprétées par le moteur PHP.

Commentaires

- Un script PHP se commente de cette manière :

```
<?php
//Commentaire sur une seule ligne, style C++

/*
    Ceci est un
    commentaire
    sur
    plusieurs lignes
*/

# Ceci est un commentaire style shell sur une seule ligne
?>
```

La fonction echo

- **echo** - Affiche une chaîne de caractères
- PHP inclut une balise ouvrante echo courte <?= qui est un raccourci au code plus verbeux <?php echo

Séparation des instructions

PHP requiert que les instructions soient terminées par un point-virgule à la fin de chaque instruction.

```
<?php  
  
echo 'Ceci est un test';  
echo 'Ceci est un autre test';  
  
?>  
  
<?=  
  
'Et un test final';  
  
?>
```

Variables

- Le typage des variables est **implicite en php**. Il n'est donc pas nécessaire de déclarer leur type au préalable ni même de les initialiser avant leur utilisation.
- Les identificateurs de variable sont précédés du symbole « \$ » (dollars). Exemple : \$prix.
- Les variables peuvent être de type entier (**integer**), réel (**double**), chaîne de caractères (**string**), tableau (**array**), objet (**object**), booléen (**boolean**).
- Il est possible de convertir une variable en un type primitif grâce au cast(1)
- `$str = "12"; // $str vaut la chaîne "12"`
- `$nbr = (int)$str; // $nbr vaut le nombre 12`

(1) : Le cast est une conversion de type. L'action de caster consiste en convertir une variable d'un type à un autre.

Variables

Un nom de variable valide doit respecter les règles suivantes (après le symbole \$) :

- Le nom est sensible à la casse : **\$a** et **\$A** sont deux variables distinctes.
 - Le premier caractère doit être une **lettre** ou un **souligné _**
 - À partir du deuxième caractère seul les lettres, chiffres ou soulignés sont acceptés
-
- `<? php`
 - `$var;` //nom de variable valide
 - `$Var;` //nom de variable valide
 - `$4vars;` //nom de variable invalide: commencer avec un nombre n'est pas autorisé
 - `$_var;` //nom de variable valide
 - `$état;` //nom de variable valide
 - `$éta+;` //nom de variable invalide: les caractères spéciaux comme + ne sont pas autorisés
 - `$var Vat;` //nom de variable invalide: l'espace n'est pas autorisé
 - `?>`

Variables

Les variables "**superglobales**" sont disponibles quel que soit le contexte du script.

\$GLOBALS

Référence toutes les variables disponibles dans un contexte global

\$_SERVER

Variables de serveur et d'exécution

\$_GET

Variables HTTP GET

\$_POST

Variables HTTP POST

\$_FILES

Variable de téléchargement de fichier via HTTP

\$_REQUEST

Variables de requête HTTP

\$_SESSION

Variables de session

\$_ENV

Variables d'environnement

\$_COOKIE

Cookies HTTP

Portée des variables

- la **portée des variables** fait référence à la **zone ou au contexte** dans lequel une variable est accessible ou modifiable. La portée d'une variable définit où et comment elle peut être utilisée dans un programme.
- une **variable globale** est une variable qui est déclarée en dehors de toute fonction, donc elle est accessible partout dans le code. Cependant, si tu veux utiliser une variable globale à l'intérieur d'une fonction, tu dois la déclarer à l'intérieur de cette fonction avec le mot-clé `global`.

```
$variableGlobale = 10; // Variable globale

1 reference
function maFonction(): void {
    global $variableGlobale; // Déclaration de la variable globale à l'intérieur de la fonction
    echo $variableGlobale . "\n"; // Utilisation de la variable globale
    // ou bien
    echo "variableGlobale : " . $GLOBALS["variableGlobale"];
}

maFonction()
```

Portée des variables

- Une **variable statique** en PHP est une variable qui conserve sa valeur entre les appels successifs à une fonction. Contrairement aux variables normales qui sont réinitialisées à chaque appel de fonction, **une variable statique** conserve sa valeur même après que la fonction a été exécutée et est appelée à nouveau.

Caractéristiques d'une variable statique :

- **Conservation de la valeur** : Une variable statique ne perd pas sa valeur à chaque appel de la fonction. Elle garde la même valeur jusqu'à ce qu'une nouvelle valeur soit affectée à cette variable dans un appel de fonction ultérieur.
- **Scope local** : Une variable statique est locale à la fonction dans laquelle elle est déclarée. Cela signifie qu'elle n'est pas accessible en dehors de cette fonction, contrairement à une variable globale.
- **Utilisation du mot-clé static** : Pour déclarer une variable statique dans une fonction, on utilise le mot-clé static.

Portée des variables

- Exemple d'utilisation du mot-clé **static**

```
function compteur(): void {  
    static $count = 0; // Déclaration d'une variable statique  
    $count++; // Incrémentation de la variable à chaque appel  
    echo $count . "\n";  
}  
  
compteur();  
compteur();  
compteur();
```

Variables dynamiques

- Une **variable dynamique** prend la valeur d'une **variable** et l'utilise comme nom d'une autre variable, en utilisant le "\$\$" précédant la variable.
- Les accolades peuvent aussi être utilisées, pour clairement délimiter le nom de la propriété
- En PHP, normalement, on déclare une variable avec un nom spécifique, par exemple : **\$tomate = "prixTomate";**
- Mais avec les variables dynamiques, tu peux utiliser le contenu d'une variable pour créer le nom d'une autre variable. Cela signifie que tu peux avoir une variable dont le nom est déterminé dynamiquement.

Syntaxe

- **\${\$var} = valeur;**
- **\$var** est une variable dont la valeur sera utilisée comme nom de la nouvelle variable.
- **\${\$var}** fait référence à cette nouvelle variable dont le nom est la valeur contenue dans \$var.

```
$tomate = "prixTomate"; // La variable $tomate contient le nom de la variable que nous voulons créer
${$tomate} = 5;         // Crée une variable nommée $prixTomate et lui assigne la valeur 5

echo $prixTomate;
```

Exemples fonctions de gestion des variables

- **empty(\$var)** : renvoie vrai si la variable est vide
- **isset(\$var)** : renvoie vrai si la variable existe
- **unset(\$var)** : détruit une variable
- **gettype(\$var)** : retourne le type de la variable
- **settype(\$var, "type")** : convertit la variable en type type (cast)
- **var_dump(\$var)**: Affiche les informations d'une variable
- **is_object(\$var)**: Détermine si une variable est de type objet

Constantes

- L'utilisateur peut définir des constantes dont la valeur est fixée une fois pour toute. Les constantes ne portent pas le symbole \$ (dollars) en début d'identificateur et ne sont pas modifiables.
- Les constantes peuvent être définies en utilisant le mot clé **const** ou en utilisant la fonction **define()**.

```
const MA_CONSTANTE = 100;  
define(constant_name: "MA_CONSTANTE", value: 1000);
```

- Contrairement aux constantes définies en utilisant l'instruction `define()`, les constantes définies en utilisant le mot-clé `const` doivent être déclarées au plus haut niveau du contexte, car elles seront définies au moment de la compilation.

Constantes magiques

Une constante magique est une constante prédéfinie dans PHP qui se change en fonction du contexte.

Nom	Description
<code>__LINE__</code>	La ligne courante dans le fichier.
<code>__FILE__</code>	Le chemin complet et le nom du fichier courant avec les liens symboliques résolus. Si utilisé pour une inclusion, le nom du fichier inclus est retourné.
<code>__DIR__</code>	Le dossier du fichier. Si utilisé dans une inclusion, le dossier du fichier inclus sera retourné. C'est l'équivalent de <code>dirname(__FILE__)</code> . Ce nom de dossier ne contiendra pas de slash final, sauf si c'est le dossier racine.
<code>__FUNCTION__</code>	Le nom de la fonction, ou {closure} pour les fonctions anonymes.
<code>__CLASS__</code>	Le nom de la classe courante. Le nom de la classe contient l'espace de nom dans lequel cette classe a été déclarée. Lorsqu'elle est utilisée dans une méthode de trait, <code>__CLASS__</code> est le nom de la classe dans laquelle le trait est utilisé.
<code>__TRAIT__</code>	Le nom du trait. Le nom du trait inclut l'espace de nom dans lequel il a été déclaré.
<code>__METHOD__</code>	Le nom de la méthode courante.
<code>__NAMESPACE__</code>	Le nom de l'espace de noms courant.

Références

- On peut à la manière des pointeurs en C faire référence à une variable grâce à l'opérateur & (ET commercial).

```
$a = 100; // la variable $a est initialisée à la valeur 100
$b = &$a; // la variable $b fait référence à $a
$a++; // on change la valeur de $a
echo $b; // qui est répercutée sur $b qui vaut alors 101
```

Les opérateurs d'affectation

- L'opérateur d'affectation le plus simple est le signe `=`.
- Il signifie que l'opérande de gauche se voit affecter la valeur de l'expression qui est à droite du signe égal.
- L'affectation par référence se fait grâce au signe `&`
- Quelle est la valeur de `$e` et `$d` après l'exécution du script ?

```
$a = 2;
$b = 3;
$c = $a + $b;
$d = &$c; // $d est une référence à $c

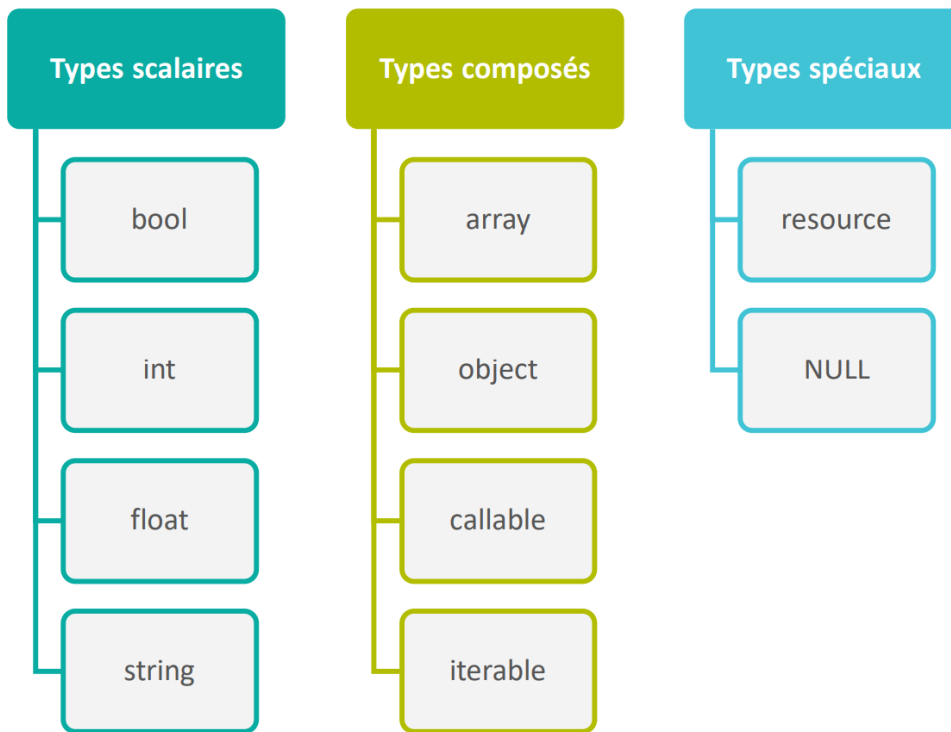
$e = "Bonjour ";
$e .= " stagiaires ISGI";

$c += $a * 2;

// Affichage des résultats pour vérification
echo "c = $c\n";
echo "d = $d\n";
echo "e = $e\n";
```

les types de données

- Les variables PHP peuvent stocker différents types de valeurs comme des nombres ou des tableaux. Par abus de langage, on parle des "types variables" de PHP.



Type booléen

- Il peut avoir true ou false.
- `==` est un opérateur qui teste l'égalité et retourne un booléen.
- Pour convertir explicitement une valeur en booléen, utilisez (**bool**) ou (**boolean**).

Lors d'une conversion en booléen, les valeurs suivantes sont considérées comme false :

- false
- l'entier 0, les nombres à virgule flottante 0.0 et -0.0
- la chaîne vide, et la chaîne "0"
- un tableau avec aucun élément
- le type spécial NULL (incluant les variables non définies)
- les objets SimpleXML créés depuis des éléments vide sans attributs.

```
if(0) echo 1;
if("") echo 2;
if("0") echo 3;
if("00") echo 4;
if("0") echo 5;
if(" ") echo 6;
```

Type booléen

- Que renvoie le programme suivant après son exécution ?
- **Var_dump** : Affiche des informations détaillées sur une variable, y compris son type et sa valeur. Il est principalement utilisé pour le débogage et l'inspection des variables.
- **Var_export** : Affiche ou retourne une représentation sous forme de chaîne d'une variable qui peut être utilisée comme du code PHP valide. Cela est utile lorsque vous souhaitez capturer la structure exacte d'une variable pour une utilisation ultérieure ou pour le débogage.
- **Json_encode**: Convertit une variable PHP (comme un tableau ou un objet) en une chaîne au format JSON. Elle est souvent utilisée pour l'échange de données entre un serveur et un client (par exemple, dans les API).

```
$bool_val = (bool>true;  
echo $bool_val;  
  
$bool_val2 = (bool>false;  
echo $bool_val2;  
  
$bool_exp1 = (bool>false;  
echo $bool_exp1 ? 'true' : 'false';  
  
$bool_exp2 = (bool>false;  
echo json_encode($bool_exp2);  
  
$bool_exp3 = (bool>false;  
echo var_export($bool_exp3);  
  
$bool_exp4 = (bool>false;  
echo (int)$bool_exp4;  
  
$bool_exp5 = (bool>false;  
var_dump($bool_exp5);
```

Type entier

- Un entier est un nombre appartenant à l'ensemble $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$.
- Les entiers peuvent être spécifiés en notation:
 - **Décimale** (base 10)
 - **Hexadécimale** (base 16) : Pour utiliser la notation hexadécimale, précédez le nombre de 0x
 - **Octale** (base 8) : Pour utiliser cette notation, précédez le nombre d'un 0 (zéro). À partir PHP 8.1.0, la notation octale peut être précédé avec 0o ou 0O
 - **Binaire** (base 2) : Pour utiliser la notation binaire, précédez le nombre de 0b.

```
<?php
// un nombre décimal
$a = 1234;
// un nombre octal (équivalent à 83 en décimal)
$a = 0123;
// un nombre octal (à partir de PHP 8.1.0)
$a = 0o123;
// un nombre hexadécimal (équivalent à 26 en décimal)
$a = 0x1A;
// un nombre binaire (équivalent à 255 en decimal)
$a = 0b11111111;
// un nombre décimal (à partir de PHP 7.4.0)
$a = 1_234_567;
?>
```

Type entier

- À partir de PHP 7.4.0, les entiers littéraux peuvent contenir des underscores (_) entre les chiffres, pour une meilleure lisibilité des littéraux. Ces underscores sont supprimés par le scanner de PHP. Exemple : `$a = 1_203_489` // c'est la même chose que `$a = 1203489`

Pour convertir explicitement une valeur en un entier, utiliser soit le mot-clé (**int**), soit (**integer**).

- Depuis un booléen : `false` correspond à 0, et `true` correspond à 1.
- Depuis un nombre décimal : le nombre sera arrondi vers zéro

Type nombre à virgules

- Le type « nombre décimal » ou **Float** en anglais.
- Pour convertir explicitement une valeur en un float, utiliser le mot-clé (float).
- Depuis une chaîne de caractères : Si une chaîne est numérique ou numérique de tête alors elle sera transformée en sa valeur flottante correspondante, sinon elle sera convertie à zéro.
- Depuis un booléen : false correspond à 0.0, et true correspond à 1.0
- Depuis d'autres types : la conversion est effectuée d'abord en int puis en float

```
// nombre Float
echo $valeur1 = (float) "100.50";
echo $valeur2 = (float) "abc";
echo $valeur3 = (float) true;
echo $valeur4 = (float) false;
echo $valeur5 = (float) 123;
```


Type chaîne de caractères

- Une variable chaîne de caractères n'est pas limitée en nombre de caractères. Elle est toujours délimitée par des simples quotes ou des doubles quotes.
- **Exemples:**

```
<?php
$nom = "ali";
$prenom = 'ali';
?>
```

Les **doubles quotes** permettent l'évaluation des variables et caractères spéciaux contenus dans la chaîne alors que les simples ne le permettent pas

- **Exemples:**

```
<?php
echo "Nom: $nom"; //affiche Nom: ali
echo 'Nom: $nom'; //afficher Nom: $nom
?>
```

Type chaîne de caractères

- Quelques caractères spéciaux :
 - `\n` : Fin de ligne (LF ou 0x0A (10) en ASCII)
 - `\r` : Retour à la ligne (CR ou 0x0D (13) en ASCII)
 - `\t` : Tabulation horizontale (HT or 0x09 (9) en ASCII)
 - `\v` : Tabulation verticale (VT ou 0x0B (11) en ASCII)
 - `\e` : échappement (ESC or 0x1B (27) en ASCII)
 - `\f` : Saut de page (FF ou 0x0C (12) en ASCII)
 - `\\` : Antislash
 - `\$` : Signe dollar
 - `\"` : Guillemet double

Type chaîne de caractères

Opérateur de **concaténation** de chaînes : . (point)

```
<?php
```

```
// Exemple 1 :
```

```
$foo = "Hello";
```

```
$bar = "World";
```

```
echo $foo.$bar;
```

```
// Exemple 2 :
```

```
$name = "ahmed";
```

```
$whoiam = $name. " ali";
```

```
//Exemple 3 :
```

```
$out = "Bonjour ";
```

```
$out .= "les stagiaires";
```

```
?>
```

Type chaîne de caractères

Quelques fonctions pour les chaînes de caractères :

- **strlen(\$str)** : retourne le nombre de caractères d'une chaîne
- **strtolower(\$str)** : conversion en minuscules
- **strtoupper(\$str)** : conversion en majuscules
- **trim(\$str)** : suppression des espaces de début et de fin de chaîne
- **substr(\$str,\$i,\$j)** : retourne une sous chaîne de \$str de taille \$j et débutant à la position \$i
- **strnatcmp(\$str1,\$str2)** : comparaison de 2 chaînes
- **addslashes(\$str)** : désécialise les caractères spéciaux (' , " , \)
- **ord(\$char)** : retourne la valeur ASCII du caractère \$char

Type chaîne de caractères

On peut délimiter les chaînes de caractères avec la syntaxe **Here-doc**.

```
$nom = "Omar";  
$essai = <<<Montext  
Bonjour $nom, comment ça va ?  
Montext;  
echo $essai;
```

La syntaxe Here-doc permet de créer des chaînes de caractères multi-lignes en PHP. Elle commence par <<< suivi d'un identifiant (un nom libre que tu choisis) et se termine par cet identifiant en nouvelle ligne, sans indentation.

La syntaxe **Here-doc** est une manière simple et propre de travailler avec des chaînes longues en PHP, particulièrement utile lorsque tu as besoin d'inclure plusieurs lignes de texte ou d'afficher du code avec une mise en forme plus complexe. Elle est aussi plus lisible que de concaténer plusieurs morceaux de texte avec des guillemets.

Type chaîne de caractères

Les fonctions d'affichage :

echo() est une fonction d'affichage la plus courante en PHP. Elle permet d'envoyer une ou plusieurs chaînes de caractères dans le navigateur.

Elle ne retourne rien (sa valeur de retour est void). Elle peut accepter plusieurs paramètres séparés par des virgules.

print() fonctionne de manière similaire à echo(), mais elle est légèrement plus lente et retourne toujours 1. Cela signifie que tu peux utiliser print() dans des expressions si nécessaire. Contrairement à echo, tu ne peux passer qu'un seul argument à print().

printf([\$format, \$arg1, \$arg2]) : la chaîne de caractère est constante et contient le format d'affichage des variables passées en argument.

Exemples :

```
$name = "Alice"; $age = 20;
```

```
echo "Bonjour, ", $name, "!";
```

```
print("Bonjour $name");
```

```
printf("Bonjour, %s. Tu as %d ans.", $name, $age);
```

Type tableau :

- Une variable tableau est de type array. Un tableau accepte des éléments de tout type. Les éléments d'un tableau peuvent être de types différents et sont séparés d'une virgule.
- Syntaxe d'un tableau est :
 - clé => valeur
 - La clé peut être soit un int, soit une chaîne de caractères. Les nombres à virgule flottante seront aussi modifiés en entier.
 - Structure de langage **array()**. La syntaxe courte remplace array() par **[]**

```
<?php
// signifie la 0 => Salut, 1 => Aloha, 2=>Hello
$tableau1 = array("Salut", "Aloha", "Hello");
Echo $tableau1[0]; // Afficher : Salut
// signifie la 0 => Salut, 1 => Aloha, 2=>Hello
$tableau2 = ["Salut", "Aloha", "Hello"];
Echo $tableau2[0]; // Afficher : Salut
$tableau3 = array(
    "0" => "Salut" ,
    "1" => "Aloha" ,
    "2" => "Hello" ,
);
Echo $tableau3[2]; // Afficher : Hello
$tableau4 = [
    "0" => "Salut" ,
    "1" => "Aloha" ,
    "2" => "Hello" ,
];
Echo $tableau4[2]; // Afficher : Hello

?>
```

Type tableau

- Quelques fonctions:
 - **count(\$tab), sizeof** : retournent le nombre d'éléments du tableau
 - **in_array(\$var,\$tab)** : dit si la valeur de **\$var** existe dans le tableau **\$tab**
 - **range(\$i,\$j)** : génère un tableau contenant les valeurs entre **\$i** et **\$j**
 - **shuffle(\$tab)** : mélange les éléments d'un tableau
 - **sort(\$tab)** : trie alphanumérique les éléments du tableau
 - **rsort(\$tab)** : trie alphanumérique inverse les éléments du tableau
 - **implode(\$str,\$tab), join** : retournent une chaîne de caractères contenant les éléments du tableau **\$tab** joints par la chaîne de jointure **\$str**

Tableaux associatifs

- Un tableau associatif est appelé aussi dictionnaire ou hashtable. permet d'associer des **clés** (qui sont généralement des chaînes de caractères) à des **valeurs**. C'est un moyen très pratique de stocker et récupérer des données où les indices ne sont pas nécessairement des entiers, mais des valeurs significatives comme des noms de champs ou des identifiants.
- L'initialisation d'un tableau associatif est similaire à celle d'un tableau normal.

- **Exemple 1**

```
$personne = array("Nom" => 'ali', "Prénom" => "khalid");
```

- **Exemple 2**

```
$personne["Nom"] = "ali";
```

```
$personne["Prénom"] = "khalid";
```

Ici à la clé "**Nom**" est associée la valeur "**Ali**".

Tableaux associatifs

- Parcours d'un tableau associatif.

```
foreach($personne as $elem) {  
    echo $elem;  
}
```

- Ici on accède directement aux **éléments du tableau** sans passer par les **clés**

```
foreach($personne as $key => $elem) {  
    echo "$key : $elem";  
}
```

Ici on accède simultanément aux clés et aux éléments.

Tableaux associatifs

Quelques fonctions :

- **array_count_values(\$tab)** : retourne un tableau contenant les valeurs du tableau **\$tab** comme clés et leur fréquence comme valeur (utile pour évaluer les redondances)
- **array_keys(\$tab)** : retourne un tableau contenant les clés du tableau associatif **\$tab**
- **array_values(\$tab)** : retourne un tableau contenant les valeurs du tableau associatif **\$tab**
- **array_search(\$val, \$tab)** : retourne la clé associée à la valeur **\$val**

Opérateurs de tableaux

1 Union (\$a + \$b)

L'opérateur + fusionne deux tableaux en conservant les clés et valeurs du premier tableau lorsque des clés identiques existent.

Exemple

```
$a = ["id" => 1, "nom" => "Alice"];  
$b = ["nom" => "Bob", "age" => 25];  
$resultat = $a + $b;  
print_r($resultat);
```

Résultat

```
Array (  
    [id] => 1  
    [nom] => Alice // "nom" vient de $a, donc "Bob" est ignoré  
    [age] => 25  
)
```

Opérateurs de tableaux

1 Union (\$a + \$b)

L'opérateur + fusionne deux tableaux en conservant les clés et valeurs du premier tableau lorsque des clés identiques existent.

Exemple

```
$a = ["id" => 1, "nom" => "Alice"];  
$b = ["nom" => "Bob", "age" => 25];  
$resultat = $a + $b;  
print_r($resultat);
```

Résultat

```
Array (  
    [id] => 1  
    [nom] => Alice // "nom" vient de $a, donc "Bob" est ignoré  
    [age] => 25  
)
```

Opérateurs de tableaux

2 Égalité (\$a == \$b)

Renvoie **true** si **\$a** et **\$b** contiennent les mêmes paires clé/valeur, peu importe l'ordre.

◆ Exemple

```
$a = ["nom" => "Alice", "age" => 25];  
$b = ["age" => 25, "nom" => "Alice"];  
var_dump($a == $b);
```

◆ Résultat

```
bool(true) // L'ordre n'a pas d'importance
```

Opérateurs de tableaux

3 Identité (\$a === \$b)

Renvoie **true** si **\$a** et **\$b** ont les mêmes clés et valeurs, dans le même ordre et du même type.

◆ Exemple

```
$a = ["nom" => "Alice", "age" => "25"]; // "25" est une chaîne  
$b = ["nom" => "Alice", "age" => 25];  // 25 est un entier  
var_dump($a === $b);
```

◆ Résultat

```
bool(false) // Les types sont différents ("25" string vs 25 int)
```

Opérateurs de tableaux

4 Inégalité (\$a != \$b ou \$a <> \$b)

Renvoie **true** si **\$a** et **\$b** n'ont pas les mêmes clés/valeurs (équivalent de == mais inversé).

◆ Exemple

```
$a = ["nom" => "Alice", "age" => 25];  
$b = ["nom" => "Bob", "age" => 25];  
var_dump($a != $b);
```

◆ Résultat

```
bool(true) // "Alice" ≠ "Bob"
```

⚠ != et <> sont identiques en PHP.

Opérateurs de tableaux

4 Non-identité (\$a !== \$b)

Renvoie **true** si **\$a** et **\$b** sont différents en valeurs, en ordre ou en type.

◆ Exemple :

```
$a = ["nom" => "Alice", "age" => 25];  
$b = ["nom" => "Alice", "age" => "25"]; // "25" en string  
var_dump($a !== $b);
```

◆ Résultat

```
bool(true) // Différence de type sur "age«
```

Type objet

- Pour créer un nouvel objet, le mot clé new est utilisé afin d'instancier une class.
- Une class définit le comportement d'un objet. La classe ne contient aucune donnée.
- Un objet est une instance d'une classe qui contient des données.
- Un membre est une variable qui appartient à un objet.
- Une méthode est une fonction qui appartient à un objet et a accès à ses membres.
- Un constructeur est une méthode spécifique qui est exécutée lorsqu'un objet est créé.

Type callable

- Les **fonctions de rappel** (ou *callback functions*) en PHP sont des fonctions qui sont passées en tant qu'arguments à une autre fonction et qui seront exécutées à un moment donné pendant l'exécution de cette fonction.
- Les fonctions de rappel peuvent être identifiées via le type callable.

Types de fonctions de rappel :

- **Fonctions simples** : Ce sont des fonctions définies dans le code. Elles sont passées par leur nom en tant qu'argument à une autre fonction.
- **Méthodes d'objet** : Les méthodes d'un objet peuvent aussi être utilisées comme callback. On les passe sous forme d'un tableau où l'index 0 contient l'objet, et l'index 1 contient le nom de la méthode.
- **Méthodes de classe statiques** : Les méthodes statiques peuvent être passées en utilisant le nom de la classe suivi de :: et du nom de la méthode (par exemple, NomClass::NomMethode).

Type callable

- Exemple d'une fonction de rappel avec une fonction simple :

```
function direBonjour($nom) {  
    return "Bonjour " . $nom;  
}  
  
function afficherMessage($callback, $nom) {  
    echo $callback($nom);  
}
```

```
afficherMessage('direBonjour', 'Ali'); // Affiche "Bonjour Ali"
```

- Exemple d'une fonction de rappel avec une méthode d'objet

```
class Personne {  
    public function direBonjour($nom) {  
        return "Bonjour " . $nom;  
    }  
}  
  
$personne = new Personne(); // Création de l'objet  
  
function afficherMessage($callback, $nom) {  
    echo $callback($nom); // Appel de la méthode via la fonction de rappel  
}  
  
// Utilisation de la méthode d'objet comme fonction de rappel  
// Le tableau contient l'objet à l'index 0 et la méthode à l'index 1  
afficherMessage([$personne, 'direBonjour'], 'Ali'); // Affiche "Bonjour Ali"
```

- Exemple d'une fonction de rappel avec une méthode de classe statique :

```
class Salutation {  
    public static function direBonjour($nom) {  
        return "Bonjour " . $nom;  
    }  
}  
  
function afficherMessage($callback, $nom) {  
    echo $callback($nom);  
}
```

```
// Utilisation d'une méthode statique de classe comme fonction de rappel avec  
afficherMessage('Salutation::direBonjour', 'Ali'); // Affiche "Bonjour Ali"
```

Type itérable

- Un **iterable** est un pseudo-type introduit en PHP 7.1. Il accepte n'importe quel tableau ou objet implémentant l'interface Traversable. (Interface permettant de détecter si une classe peut être parcourue en utilisant **foreach**).
- Tous les tableaux sont des itérables.
- Un tableau peut être utilisé comme argument d'une fonction qui nécessite un itérable.

Un itérateur doit avoir ces méthodes :

- **current()** : Renvoie l'élément sur lequel le pointeur est entrain de pointer.
- **key()** : Renvoie la clé associée à l'élément courant dans la liste.
- **next()** : Déplace le pointeur vers l'élément suivant dans la liste.
- **rewind()** : Déplace le pointeur sur le premier élément de la liste.
- **valid()** : Il doit retourner false si le pointeur interne ne pointe sur rien (par exemple, si next() est appelé à la fin de la liste). Dans tous les autres cas, il retourne vrai.

Modification de types

(int), (integer) :
modification en int

(bool), (boolean) :
modification en bool

(float), (double), (real) :
modification en float

(string) :
modification en string

(array) :
modification en array

(object) :
modification en object

Contrôles de flux et boucles

- L'ordre de flux est l'ordre dans lequel les instructions d'un programme sont exécutées.
- Les instructions qui changent l'ordre d'exécution sont :
 - Les itérations.
 - Les instructions conditionnelles.
 - Les ruptures de séquences
- Les instructions de contrôle sont utilisées pour contrôler l'exécution d'un programme :
 - **Instructions de boucle** : elles sont utilisées pour exécuter un bloc de code à x reprises.
 - **Instructions de sélection** : elles vous permettent d'exécuter un bloc de code spécifique dans plusieurs blocs de code en fonction de critères de sélection.
 - **Instructions de saut** : Ces instructions sont utilisées pour passer d'un bloc de code à un autre.

Syntaxe alternative

- En PHP, pour les structures de contrôle telles que if, while, for, foreach et switch, il existe une autre manière de regrouper des instructions à l'intérieur d'un bloc en remplaçant les accolades d'ouverture par un deux-points (:) et les accolades de fermeture par endif;, endwhile;, endfor;, endforeach; ou endswitch; selon la structure utilisée. Cependant, il est important de noter que vous ne pouvez pas mélanger ces deux syntaxes (avec ou sans accolades) dans un même bloc de contrôle.

```
<?php

$age = 20;

if ($age ≥ 18) :
    echo "Vous êtes majeur.";
else :
    echo "Vous êtes mineur.";
endif;

?>
```


L'instruction if else elseif

L'instruction if else

- Si l'expression vaut true, PHP exécutera l'instruction et si elle vaut false, l'instruction sera ignorée.
- Les instructions après le else ne sont exécutées que si l'expression du if est false.

L'instruction elseif

- L'expression elseif est exécutée seulement si le if précédent et tout autre elseif précédent sont évalués comme false, et que votre elseif est évalué à true.

```
<?php

if (expression):
    commandes
else:
    commandes
endif;

?>
```

```
<?php
$age = 20;
if ($age < 13) {
    echo "Vous êtes un enfant.";
} elseif ($age ≥ 13 && $age < 18) {
    echo "Vous êtes un adolescent.";
} elseif ($age ≥ 18 && $age < 65) {
    echo "Vous êtes un adulte.";
} else {
    echo "Vous êtes un senior.";
}

?>
```

L'instruction switch

- L'instruction switch équivaut à une série d'instructions if.
- Un cas spécial est default. Ce cas est utilisé lorsque tous les autres cas ont échoué.
- PHP continue d'exécuter les instructions jusqu'à la fin du bloc d'instructions du switch, ou bien dès qu'il trouve l'instruction break.
- Dans une commande switch, une condition n'est évaluée qu'une fois, et le résultat est comparé à chaque case.

```
<?php
switch ($i) {
case 0:
    echo "i égal 0";
    break;
case 1:
    echo "i égal 1";
    break;
case 2:
    echo "i égal 2";
    break;
default:
    echo "i n'est ni égal à 2, ni à 1, ni à 0.";
}
?>
```

L'instruction match

- De la même manière qu'une instruction **switch**, l'instruction **match** a une expression de sujet qui est comparée à plusieurs alternatives.
- Différences entre match et switch :
 - match évaluera une valeur un peu comme les expressions ternaires.
 - la comparaison match est un contrôle d'identité (===) plutôt qu'un contrôle d'égalité faible (==) comme switch.
 - match renvoie une valeur

```
<?php
$jour = 3;
$resultat = match ($jour) {
    1 => "Lundi",
    2 => "Mardi",
    3 => "Mercredi",
    4 => "Jeudi",
    5 => "Vendredi",
    6 => "Samedi",
    7 => "Dimanche",
    default => "Jour invalide",
};
echo $resultat;
?>
```

L'instruction while

- PHP exécute l'instruction tant que l'expression de la boucle **while** est évaluée comme **true**. Si l'expression du **while** est false avant la première itération, l'instruction ne sera jamais exécutée

```
<?php
$i = 1;
while ($i ≤ 5) :
    echo "Itération n°: $i \n";
    $i++;
endwhile;
?>
```

L'instruction do-while

- La principale différence par rapport à la boucle **while** est que la première itération de la boucle **do-while** est toujours exécutée.

```
<?php
$i = 1;
do :
    echo "Itération n°: $i \n";
    $i++;
while ($i ≤ 5);
?>
```

L'instruction for

- L'instruction for est utilisée pour exécuter un bloc de code un nombre spécifique de fois. Elle est souvent utilisée lorsqu'on connaît à l'avance combien de fois une boucle doit s'exécuter.

Syntaxe

for (initialisation; condition; incrémentation) :

// Code à répéter

endfor;

- Initialisation : Déclare et initialise une variable avant le début de la boucle.
- Condition : La condition qui doit être vraie pour que la boucle continue. Si elle devient fausse, la boucle s'arrête.
- Incrémentation : Modifie la variable d'itération à chaque tour (souvent utilisée pour avancer ou reculer dans un tableau, ou itérer sur des nombres).

```
<?php
for ($i = 1; $i ≤ 5; $i++) :
    echo "Itération n°: $i \n";
endfor;
?>
```

L'instruction foreach

- L'instruction **foreach** est utilisée pour parcourir les éléments d'un tableau ou d'un objet. Elle permet d'itérer facilement sur chaque élément d'un tableau, sans avoir à gérer manuellement un indice.

Syntaxe

foreach (\$tableau as \$clé => \$valeur) :
 // Code à exécuter pour chaque élément
endforeach;

- **\$tableau** : Le tableau ou l'objet que vous souhaitez parcourir.
- **\$clé** : L'index ou la clé de l'élément courant (optionnel si vous n'en avez pas besoin).
- **\$valeur** : La valeur de l'élément courant.

```
<?php
$personnes = array("Alice" => 25, "Bob" => 30, "Charlie" => 35);
foreach ($personnes as $nom => $age) :
    echo "$nom a $age ans. \n";
endforeach;
?>
```

L'instruction break

- L'instruction **break** est utilisée pour interrompre l'exécution d'une boucle (comme **for**, **while**, **do-while**, ou **foreach**) ou d'un switch avant que sa condition de fin ne soit atteinte. Elle permet de sortir prématurément de la structure de contrôle.

Syntaxe

- `break;`

```
<?php
$i = 1;
while ($i <= 10) :
    if ($i == 5) :
        break; // Sort de la boucle lorsque $i est égal à 5
    endif;
    echo "Valeur de i: $i \n";
    $i++;
endwhile;
?>
```


L'instruction continue

- L'instruction **continue** est utilisée pour passer à l'itération suivante d'une boucle sans exécuter le reste du code à l'intérieur de la boucle pour l'itération courante. Elle est utile lorsque vous voulez sauter certaines itérations en fonction d'une condition donnée.

Syntaxe continue;

```
<?php
for ($i = 1; $i ≤ 5; $i++) :
    if ($i = 3) :
        continue; // Passe à l'itération suivante lorsque $i est égal à 3
    endif;
    echo "Valeur de i: $i \n";
endfor;
?>
```


Fonctions

- Comme tout langage de programmation, php permet l'écriture de fonctions.
- Les fonctions peuvent prendre des arguments dont il n'est pas besoin de spécifier le type. Elles peuvent de façon optionnelle retourner une valeur.
- L'appel à une fonction peut ne pas respecter son prototypage (nombre de paramètres). Les identificateurs de fonctions sont insensibles à la casse.

```
function mafunction($a){  
    $a += 15;  
    echo "salut !!";  
    return ($a + 10);  
}  
  
$nbr = MaFunction(15.1);  
  
echo $nbr;
```

Fonctions

- On peut donner une valeur par défaut aux arguments lors de la déclaration de la fonction. Ainsi, si un argument est « oublié » lors de l'appel de la fonction, cette dernière lui donnera automatiquement une valeur par défaut décidée à l'avance par le programmeur.



```
function Set_Color($color='black') {  
    global $car;  
    $car["color"] = $color;  
}
```

- Les paramètres sont passés par copie et les résultats sont retournés par copie (sauf à forcer la référence). Même sans paramètre, un entête de fonction doit porter des parenthèses (). Les différents arguments sont séparés par une virgule , . Et le corps de la fonction est délimité par des accolades { }.

Fonctions

- Une fonction peut être définie après son appel :

```
$nbr = MAFUNCTION();

function mafunction($a = 5){
    $a += 15;
    echo "salut !!";
    return ($a + 10);
}

echo $nbr;
```

Méthode GET

- GET est une méthode **HTTP** utilisée pour demander des données depuis un serveur.
- Les informations sont envoyées dans l'URL de la requête.
- Elle est souvent utilisée pour récupérer des ressources (comme une page web ou une image).

Caractéristiques principales de GET :

- **Visibilité des données** : Les données envoyées (paramètres) sont visibles dans l'URL.
- **Idéale pour la récupération d'informations** : Utilisée pour obtenir des informations sans changer l'état du serveur.
- **Non sécurisé** : Les données sont visibles dans l'URL, donc ne pas l'utiliser pour des informations sensibles.

Exemple d'URL GET

- `https://www.example.com/search?query=chat`

Méthode POST

- POST est une méthode HTTP utilisée pour envoyer des données au serveur.
- Les données sont envoyées dans le corps de la requête, et non dans l'URL.
- Utilisée pour créer ou modifier des ressources sur le serveur (par exemple, un formulaire).

Caractéristiques principales de POST :

- **Sécurisé** : Les données ne sont pas visibles dans l'URL, ce qui est plus sécurisé que GET.
- **Idéale pour l'envoi de données sensibles** : Utilisée pour l'envoi de formulaires ou de données confidentielles.

Formulaires simples

- Lorsqu'un **formulaire** est envoyé à un script PHP, toutes les variables du formulaire seront automatiquement disponibles dans le script.
- Exemple d'un formulaire qui demande de remplir le nom et prénom avec un bouton OK. Lorsque un visiteur remplit le formulaire, et clique sur le bouton OK, le fichier **action.php** est appelé :

```
<form action="action.php" method="post">
  <p>Votre nom : <input type="text" name="nom" /> </p>
  <p>Votre prénom : <input type="text" name="prenom" /> </p>
  <p><input type="submit" value="OK"> </p>
</form>
```

\$_GET

- Elle donne les valeurs des informations indiquées dans l'url.
- Les informations après le point d'interrogation ? d'une URL sont en réalité des données que l'on fait transiter d'une page à une autre.
- **Exemple :**
 - Le site s'appelle : **lesite.com**
 - Ma page PHP de utilisateur : **action.php**
 - Pour accéder à la page de l'utilisateur, l'URL sera : **http://www.lesite.com/action.php**
 - Pour envoyer les informations de l'utilisateur à la page php nous utiliserons :
http://www.lesite.com/action.php?nom=Rasmus&prenom=Lerdorf
- Les paramètres sont séparés par le symbole **&**
- Le fichier « action.php » peut employer la variable super globale **\$_GET** pour récupérer les données du formulaire.

```
<?php
    echo "Bonjour $_GET["nom"] $_GET["prenom"] et bienvenue";
?>
```


\$_POST et \$_REQUEST

- PHP **\$_POST** est une variable super globale qui est utilisée pour collecter des données de formulaire après avoir soumis un formulaire HTML avec **method="post"**.
- \$_POST est également utilisé pour passer des variables
- Lorsque nous envoyons des variables via un formulaire, la variable est récupérée de cette façon **\$_POST['Variable']** et non plus **\$Variable**

Exemple de formulaire avec la méthode POST

```
<form action="action.php" method="post">
  <p>Votre nom : <input type="text" name="nom" /> </p>
  <p>Votre prénom : <input type="text" name="prenom" /> </p>
  <p><input type="submit" value="OK"></p>
</form>
```

- Pour accéder aux variables du formulaire :

```
<?php
echo $_POST['nom'];
echo $_REQUEST['prenom'];
?>
```

Variables d'environnement

`$_SERVER`

- `$_SERVER` est un tableau contenant des informations comme les en-têtes, dossiers et chemins du script.

`PHP_SELF`

Le nom du fichier du script en cours d'exécution, par rapport à la racine web

`GATEWAY_INTERFACE`

Numéro de révision de l'interface CGI du serveur

`SERVER_ADDR`

L'adresse IP du serveur sous lequel le script courant est en train d'être exécuté

`SERVER_NAME`

Le nom du serveur hôte qui exécute le script suivant

`SERVER_SOFTWARE`

Chaîne d'identification du serveur, qui est donnée dans les en-têtes lors de la réponse aux requêtes

`REQUEST_METHOD`

Méthode de requête utilisée pour accéder à la page

`HTTP_ACCEPT_LANGUAGE`

Contenu de l'en-tête Accept-Language: de la requête courante, si elle existe

`HTTP_CONNECTION`

Contenu de l'en-tête Connection: de la requête courante, si elle existe

`HTTP_HOST`

Contenu de l'en-tête Host: de la requête courante, si elle existe

`HTTPS`

Défini à une valeur non-vide si le script a été appelé via le protocole HTTPS

`REMOTE_PORT`

Le port utilisé par la machine cliente pour communiquer avec le serveur web

`REMOTE_USER`

L'utilisateur authentifié

Variables d'environnement

Créer une page PHP nommée **infos_serveur.php** qui affiche dans une structure claire (tableau HTML ou liste) les informations suivantes :

- Le chemin du script (PHP_SELF)
- L'adresse IP du serveur (SERVER_ADDR)
- Le nom du serveur (SERVER_NAME)
- Le logiciel serveur (SERVER_SOFTWARE)
- L'interface CGI (GATEWAY_INTERFACE)
- La méthode de la requête (REQUEST_METHOD)
- Le protocole sécurisé HTTPS (HTTPS)
- Le port utilisé par le client (REMOTE_PORT)
- L'utilisateur authentifié (REMOTE_USER)
- Le contenu des en-têtes Host, Connection et Accept-Language

Variables d'environnement

\$_FILES - Variable de téléchargement de fichier via HTTP :

- **\$_FILES['nom_de_la_variable']['name']** : Le nom original du fichier qui provient de la machine du client
- **\$_FILES['nom_de_la_variable']['type']** : Le type MIME du fichier.
- **\$_FILES['nom_de_la_variable']['size']** : La taille du fichier en bytes.
- **\$_FILES['nom_de_la_variable']['tmp_name']** : Le nom temporaire généré et stocké sur le serveur (dans un dossier temporaire)
- **\$_FILES['nom_de_la_variable']['error']** : Le code erreur associé à l'upload, (0 = succès, 1-7 = erreurs)

\$_SESSION - Variables de session. Liste des fonctions dans

<https://www.php.net/manual/fr/ref.session.php>

\$_ENV - Variables d'environnement

\$_COOKIE - Cookies HTTP

La fonction `header()` en PHP

La fonction `header()` en PHP permet d'envoyer des en-têtes HTTP au navigateur avant tout affichage (HTML ou echo). Les en-têtes sont des instructions envoyées par le serveur au navigateur pour lui dire comment se comporter. Elle est utile pour rediriger les utilisateurs, définir le type de contenu, forcer le téléchargement de fichiers, etc.

Syntaxe et paramètres :

```
header(string $header, bool $replace = true, int $response_code = 0);
```

- **\$header (obligatoire)** : La chaîne contenant l'en-tête HTTP à envoyer.
- **\$replace (optionnel, par défaut true)** : Détermine si l'en-tête doit remplacer un en-tête existant du même type.
- **\$response_code (optionnel)** : Définit le code HTTP à envoyer (ex : 301, 404, etc.).

La fonction header() en PHP

Exemples d'utilisation de header() :

- Redirection vers une autre page

```
<?php
```

```
    header("Location: https://www.exemple.com");  
    exit(); // Toujours utiliser exit() après une redirection
```

```
?>
```

Cela redirige l'utilisateur vers une autre page.

- Définir le type de contenu

Si vous envoyez des données JSON, vous devez définir le bon type MIME :

```
<?php
```

```
    header("Content-Type: application/json");  
    echo json_encode(["message" => "Bonjour !"]);
```

```
?>
```

La fonction header() en PHP

Exemples d'utilisation de header() :

- Retourner une erreur 404 personnalisée

```
<?php
    header($_SERVER["SERVER_PROTOCOL"] . " 404 Not Found");
    echo "Page non trouvée";
    exit();
?>
```

Remarque importante :

header() doit être appelée avant tout affichage HTML ou echo

Sinon → Erreur : "headers already sent"

La fonction header() en PHP

**Créer un formulaire de connexion contenant deux champs :
login et mot de passe.**

**Le traitement sera effectué via PHP, avec une redirection vers
une page de succès ou d'erreur selon les données saisies.**

Fonctions sur les Date/Heure

Checkdate()

La fonction checkdate() permet de vérifier si une date donnée est valide ou non.

checkdate(int \$month, int \$day, int \$year): bool

- **\$month** : Le mois (entre 1 et 12).
- **\$day** : Le jour (doit être valide pour le mois donné).
- **\$year** : L'année (doit être un nombre entier positif). Retourne true si la date est valide, sinon false.

<?php

```
    if (checkdate(2, 28, 2025)) {  
        echo "Date valide";  
    } else {  
        echo "Date invalide";  
    }
```

?>

Fonctions sur les Date/Heure

`DateTime::add()`

Les fonctions **`DateTime::add()`** et **`date_add()`** permettent d'ajouter une durée (jours, mois, années, heures, etc.) à une date.

`DateTime::add()` C'est une méthode de l'objet **`DateTime`** qui modifie l'objet en ajoutant un intervalle de temps.

Syntaxe : **`DateTime::add(DateTimeInterval $interval): DateTime`**

- `$interval` : Un objet **`DateInterval`** qui définit l'intervalle de temps à ajouter.
- Retourne l'objet **`DateTime`** modifié.

```
<?php
$date = new DateTime("2024-04-01"); // Date de départ
$date->add(new DateInterval("P10D")); // Ajoute 10 jours
echo $date->format("Y-m-d"); // Affiche la nouvelle date
?>
```

Fonctions sur les Date/Heure

L'argument de **DateInterval** est une chaîne de format spécial qui définit une durée (jours, mois, années, heures, etc.). Il commence toujours par **"P"** (pour Period) et peut inclure plusieurs parties.

Syntaxe du format de DateInterval

```
new DateInterval("PnYnMnDTnHnMnS")
```

- P = Début de la période (obligatoire).
- nY = Nombre d'années (Years).
- nM = Nombre de mois (Months).
- nD = Nombre de jours (Days).
- T = Séparateur pour la partie "temps" (Time, facultatif).
- nH = Nombre d'heures (Hours).
- nM = Nombre de minutes (Minutes).
- nS = Nombre de secondes (Seconds).

Exemple :

```
$interval = new DateInterval("P1Y2M15DT5H20M"); // 1 an, 2 mois, 15 jours, 5 heures, 20 minutes
```

⚠ Si tu veux ajouter des heures, minutes ou secondes, tu dois inclure "T".

⚠ Les valeurs doivent être en entier, pas de décimales.

Fonctions sur les Date/Heure

`DateTime::setDate()`

C'est une méthode de l'objet **DateTime** qui change la date tout en conservant l'heure actuelle.

Syntaxe

`DateTime::setDate(int $year, int $month, int $day): DateTime`

- **`$year`** : L'année à définir.
- **`$month`** : Le mois (1 à 12).
- **`$day`** : Le jour (1 à 31).

Exemple d'utilisation

```
<?php
$date = new DateTime("2024-04-01 15:30:00");
$date->setDate(2025, 12, 25);
echo $date->format("Y-m-d H:i:s");
?>
```

Fonctions sur les Date/Heure

DateTime::diff()

C'est une méthode de l'objet **DateTime** qui calcule la différence entre deux dates.

Syntaxe

`DateTime::diff(DateTimeInterface $datetime2, bool $absolute = false): DateInterval`

- `$datetime2` : La deuxième date à comparer.
- `$absolute` (optionnel, par défaut `false`) :
 - `false` → Différence signée (négatif si `$datetime2` est avant la date d'origine).
 - `true` → Différence absolue (toujours positive).
- Retourne un objet `DateInterval`.

```
<?php
```

```
$date1 = new DateTime("2024-04-01");
```

```
$date2 = new DateTime("2025-06-15");
```

```
$diff = $date1->diff($date2);
```

```
echo "Différence : " . $diff->y . " ans, " . $diff->m . " mois, " . $diff->d . " jours.";
```

```
?>
```

Fonctions sur les Date/Heure

DateTime::sub() et date_sub()

Ces deux fonctions sont utilisées pour soustraire un intervalle de temps d'un objet **DateTime**. Elles permettent de manipuler facilement les dates en soustrayant des périodes comme des jours, des mois, ou des années.

Syntaxe

`public DateTime::sub(DateInterval $interval): void`

`$interval` : Un objet `DateInterval` représentant l'intervalle à soustraire de la date.

`date_sub()`

`date_sub(DateTime $datetime, DateInterval $interval): void`

`$datetime` : Un objet `DateTime` à modifier.

`$interval` : Un objet `DateInterval` représentant l'intervalle à soustraire de la date.

Exemple

```
$date = new DateTime(); // Date actuelle
```

```
// Soustraction de 5 jours
```

```
$date->sub(new DateInterval('P5D'));
```

```
echo $date->format('Y-m-d');
```

```
$date = new DateTime(); // Date actuelle
```

```
// Soustraction de 1 mois
```

```
date_sub($date, new DateInterval('P1M'));
```

```
echo $date->format('Y-m-d');
```

Fonctions sur les Date/Heure

getdate()

La fonction getdate() retourne un tableau associatif contenant des informations détaillées sur la date et l'heure courantes, ou sur un timestamp donné.

timestamp : c'est un **nombre entier** qui représente le **nombre de secondes écoulées depuis le 1er janvier 1970 à minuit UTC** (appelé "l'époque Unix").

Syntaxe

getdate(int \$timestamp = time()): array

Exemple : Obtenir les informations sur la date actuelle :

```
<?php
$date_info = getdate(); // Récupère les informations sur la date actuelle
echo "Année : " . $date_info['year'] . "<br>";
echo "Mois : " . $date_info['month'] . "<br>";
echo "Jour : " . $date_info['mday'] . "<br>";
echo "Jour de la semaine : " . $date_info['weekday'] . "<br>";
echo "Heure : " . $date_info['hours'] . "<br>";
echo "Minutes : " . $date_info['minutes'] . "<br>";
?>
```

Fonctions sur les Date/Heure

Voici les clés du tableau retourné par **getdate()** :

Clé	Signification	Exemple
seconds	Secondes (0 à 59)	30
minutes	Minutes (0 à 59)	45
hours	Heures (0 à 23)	14
mday	Jour du mois (1 à 31)	21
wday	Jour de la semaine (0 = dimanche)	1
mon	Mois (1 à 12)	4
year	Année	2025
yday	Jour de l'année (0 à 365)	110
weekday	Nom du jour (en anglais)	"Monday"
month	Nom du mois (en anglais)	"April"

Fonctions sur les Date/Heure

Date()

La fonction **date()** est utilisée pour formater une date et une heure locale en PHP. Elle prend un format spécifique comme argument et retourne une chaîne de caractères représentant la date actuelle ou une date donnée.

Syntaxe

date(string \$format, int \$timestamp = time()): string

- **\$format** : Une chaîne définissant la structure de la date à retourner.
- **\$timestamp (optionnel)** : Un horodatage Unix. Par défaut, la date et l'heure actuelles sont utilisées.

```
<?php  
echo date("l, d F Y"); // Affiche le jour complet, le jour du mois et l'année  
?>
```

Fonctions sur les Date/Heure

Date()

d	Jour (2 chiffres)	01 à 31
j	Jour (sans zéro)	1 à 31
m	Mois (2 chiffres)	01 à 12
n	Mois (sans zéro)	1 à 12
Y	Année (4 chiffres)	2023
y	Année (2 chiffres)	23
H	Heure (24h)	00 à 23
i	Minutes	00 à 59
s	Secondes	00 à 59
D	Jour (texte court)	Mon, Tue...
l	Jour (texte complet)	Monday, Tuesday...
F	Mois (texte complet)	January, February...

Fonctions sur les Date/Heure

Date()

Caractères pour le paramètre format	Description	Exemple de valeurs retournées
<i>Jour</i>	---	---
d	Jour du mois, sur deux chiffres (avec un zéro initial)	01 à 31
D	Jour de la semaine, en trois lettres (et en anglais - par défaut : en anglais, ou sinon, dans la langue locale du serveur)	Mon à Sun
j	Jour du mois sans les zéros initiaux	1 à 31
l ('L' minuscule)	Jour de la semaine, textuel, version longue, en anglais	Sunday à Saturday
N	Représentation numérique ISO 8601 du jour de la semaine	1 (pour Lundi) à 7 (pour Dimanche)
S	Suffixe ordinal d'un nombre pour le jour du mois, en anglais, sur deux lettres	st, nd, rd ou th. Fonctionne bien avec j
w	Jour de la semaine au format numérique	0 (pour dimanche) à 6 (pour samedi)
z	Jour de l'année	0 à 365
<i>Semaine</i>	---	---
W	Numéro de semaine dans l'année ISO 8601, les semaines commencent le lundi	Exemple : 42 (la 42ème semaine de l'année)