PHP - Formulaires

1 - Création du formulaire HTML

Un formulaire simple permet de collecter les informations utilisateur.

Utilisez les types de champs adaptés et les attributs pour limiter les saisies (ex. : `required` , `type="email"`).

```html
<form action="traitement.php" method="get">
  <label for="nom">Nom :</label>
  <input type="text" name="nom" id="nom" required>
  <label for="email">Email :</label>
  <input type="email" name="email" id="email" required>
  <button type="submit">Envoyer</button>
</form>
```

Récupération des données côté serveur

2.1 - $_GET

La superglobale `$_GET` contient les données envoyées via la méthode GET (paramètres dans l'URL).

```php
<?php

// Récupération des données
$nom = $_GET['nom'] ?? '';
$email = $_GET['email'] ?? '';

// Affichage des données (pour vérifier)
echo "Nom: " . htmlspecialchars($nom) . "<br>";
echo "Email: " . htmlspecialchars($email) . "<br>";

?>
```

2.2 - $_POST

La superglobale $_POST contient les données envoyées via la méthode POST (dans le corps de la requête).

```php
// Formulaire avec method="post"
$nom = $_POST['nom'] ?? '';
$email = $_POST['email'] ?? '';
$message = $_POST['message'] ?? '';

// Vérifier si le formulaire a été soumis
if (!empty($_POST)) {
    // Traitement des données
}
```

2.3 - $_SERVER['PHP_SELF']

L'attribut `action` du formulaire peut utiliser `$_SERVER['PHP_SELF']` pour renvoyer les données au même script.

```php
// Formulaire auto-soumis
echo '<form action="' . $_SERVER['PHP_SELF'] . '" method="post">';

// Attention : toujours sécuriser avant affichage
$action = htmlspecialchars($_SERVER['PHP_SELF'], ENT_QUOTES, 'UTF-8');
echo '<form action="' . $action . '" method="post">';
```

2.4 - $_SERVER['REQUEST_METHOD']

La superglobale `$_SERVER['REQUEST_METHOD']` permet de connaître la méthode HTTP utilisée pour accéder à la page (GET, POST, etc.).

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  // Traitement du formulaire
}
```

2.5 - Bonnes pratiques

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  // Vérification globale de la méthode POST
  if (isset($_POST['nom'])) {
    $nom = $_POST['nom'];
    // Traitement du nom
  } else {
    // Erreur : le champ 'nom' est requis
    echo "Le champ 'nom' est requis.";
  }

  if (isset($_POST['email'])) {
    $email = $_POST['email'];
    // Traitement de l'email
  } else {
    // Le champ 'email' est optionnel, on peut continuer sans
    $email = ''; // Valeur par défaut
  }
} else {
  // Affichage du formulaire
}
```

Etapes de traitement d'un formulaire

Ordre optimal recommandé :

nettoyage → validation → sécurisation est le plus cohérent et le plus sécurisé pour traiter un formulaire .

1. Nettoyage : préparer la donnée brute.
2. Validation : vérifier que la donnée nettoyée est conforme.
3. Sécurisation : échapper la donnée avant usage risqué.

Étape	Objectif principal	À quel moment ?
Nettoyage	Normaliser la donnée brute	Dès réception
Validation	Vérifier la conformité	Après nettoyage
Sécurisation	Protéger contre les attaques	Avant affichage ou stockage

3.1 - Nettoyage

Le **nettoyage de données**, ou `data cleansing`, est l'opération qui permet de **détecter**, **corriger**, **remplacer** ou **supprimer** les données **erronées**, **incomplètes**, **non pertinentes** ou **incohérentes** afin d'assurer leur qualité, leur cohérence et leur fiabilité.

Ce processus améliore la qualité des données utilisées pour l'analyse ou la prise de décision.

Filtres natifs en PHP :

- `filter_input()` + `FILTER_SANITIZE`

Approche manuelle en PHP :

- trim() : supprime les espaces inutiles en début et fin de chaîne.
- stripslashes() : retire les antislashes ajoutés automatiquement (peut être utile si magic_quotes est activé, ce qui est rare aujourd'hui).
- strip_tags() : retire les balises HTML et PHP (optionnel selon le contexte).

Exemple natif :

```php
 $nom = filter_input(INPUT_POST, 'nom', FILTER_SANITIZE_FULL_SPECIAL_CHARS);
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
```

Exemple manuel :

```php
 function nettoyer($donnee) {
   $donnee = trim($donnee);
   $donnee = stripslashes($donnee);
   $donnee = strip_tags($donnee); // optionnel
   return $donnee;
}

$nom = nettoyer($_POST['nom'] ?? '');
$email = nettoyer($_POST['email'] ?? '');
$message = nettoyer($_POST['message'] ?? '');
```

3.2 - Validation

> La **validation de données** est le processus qui consiste à **vérifier que les données saisies** dans un système **respectent des critères** ou des **règles prédéfinis**, afin d'assurer leur exactitude, leur complétude (absence d'informations manquantes) et leur cohérence avant leur stockage ou leur traitement.
>
> Elle vise à garantir que seules les données valides et conformes soient acceptées.

Fonctions natives PHP :

- filter_input() : pour valider les types simples (email, entier, etc.).
- preg_match() : pour valider des formats personnalisés avec une expression régulière.

Exemple :

```php
 if (!filter_input($email, FILTER_VALIDATE_EMAIL)) {
   echo "L'email n'est pas valide.";
   exit;
}

// Le pseudo ne peut contenir que des lettres, des chiffres ou des underscores, faire entre 3 et 20 caractères.
if (!preg_match('/^[a-zA-Z0-9_]{3,20}$/', $pseudo)) {
   // Erreur
}
```

Quand ?

La validation doit être faite **après le nettoyage** pour s'assurer que la donnée soumise à la règle est bien celle que l'utilisateur voulait envoyer, sans éléments parasites.

3.3 - Sécurisation

> La **sécurisation des données** désigne l'ensemble des mesures, outils et procédures mis en œuvre pour **protéger les informations** numériques contre tout accès non autorisé, la corruption ou le vol, tout au long de leur cycle de vie.
>
> Elle vise à garantir la confidentialité, l'intégrité et la disponibilité des données.

Exemple :

```php
 $nom = htmlspecialchars($nom, ENT_QUOTES, 'UTF-8');
$email = htmlspecialchars($email, ENT_QUOTES, 'UTF-8');
$message = htmlspecialchars($message, ENT_QUOTES, 'UTF-8');

// Affichage
echo $nom . ' | '.$email . ' | '.$message;
```

Quand ? :

La sécurisation intervient **juste avant l'affichage** ou l'envoi dans un contexte potentiellement dangereux (HTML, base de données, etc.)

3.4 - Bilan

```php
 // 1.NETTOYAGE
function nettoyer($data) {
   $data = trim($data);
   $data = stripslashes($data);
   $donnee = strip_tags($donnee); // optionnel
   return $data;
}

$pseudo = nettoyer($_POST['pseudo'] ?? '');

// 2.VALIDATION
if (!preg_match('/^[a-zA-Z0-9_]{3,20}$/', $pseudo)) {
   // Erreur
}

// 3.SECURISATION (juste avant affichage)
$pseudo = htmlspecialchars($pseudo, ENT_QUOTES, 'UTF-8');

// 4. Affichage (une fois nettoyées, validées et sécurisées, les données peuvent être affichées ou utilisées dans un mail, un fichier,
         etc.)
echo $pseudo;
```

Résumé des fonctions clés

Fonction	Usage principal
`trim()`	Supprime les espaces inutiles
`stripslashes()`	Retire les antislashes (optionnel)
`strip_tags()`	Supprime les balises HTML/PHP (optionnel)
`htmlspecialchars()`	Sécurise l'affichage contre le XSS
`filter_input()`	Valide un type simple (email, entier, etc.)
`preg_match()`	Valide un format personnalisé avec une regex

Bonnes pratiques

- Toujours nettoyer et valider les données côté serveur.
- Ne jamais faire confiance aux données utilisateur.
- Utiliser `htmlspecialchars()` avant tout affichage pour éviter le XSS.
- Combiner nettoyage + validation + sécurisation pour une protection maximale.