PROGRAMMATION PHP-8 – MYSQL-8
ECHANGE D'INFORMATIONS PHP
$_POST
L2S – 7-9 AVRIL 2025
3 JOURNEES

http://php.net/manual/fr/langref.php
http://www.w3schools.com/php/
https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql

SOMMAIRE

Sommaire 1

Echanges d'informations en PHP 2
0. Présentation du document 2
Remarque 2
Document « slidé » 2
1. Méthode POST - Formulaires - Le tableau $_POST 3
Les formulaires de saisie en HTML 3
Exemple-4 : formulaire-POST 6
exemple 5 : formulaire-GET 7
2. Toujours vérifier les informations fournies par l'utilisateur – 2 : faille XSS 8
Never trust user input ! 8
Exemple-s 4 et 5 8
Solution : exemple-6 : htmlspecialchars() et htmlentities() 9
Fonctions de traitement de chaînes de caractères 10

TP - $_POST 11
3. TP-1 : Exercice password 11
TP-1 : Version 1 11
Remarque : crypter les mots de passe 12
TP-1 : Version 2 : monofichier 13
TP-1 : Version 3 : monofichier sur le $_GET + MVC + multi-fichiers 13

Edition : septembre 2021 – avril 2025

ECHANGES D'INFORMATIONS EN PHP

0. Présentation du document

- Les exemples sont présentés dans un chapitre en vert avec le mot clé : exemple-
- Les dossiers d'exemples sont fournis dans l'article qui contient ce fichier de cours.
  - ➔ http://bliaudet.free.fr/IMG/zip/PHP-02-Echanges_d_informations.zip
  - ➔ Chargez ce fichier et mettez-le dans le dossier « php » du répertoire web « www » du serveur WEB. Vous pouvez aussi structurer les choses avec des dossiers J1, J2, etc. correspondant aux journées de travail.

- Les exercices à faire sont présentés dans un chapitre en jaune avec les mots-clés : TP- et exercice-
  - ➔ Les sources pour les exercices, quand il y en a, sont fournis dans le dossier des exemples.

Remarque

- Certains fichiers d'exemple commencent par ces trois lignes :

```
echo '<h1>CODE PHP</h1>';
highlight_file('fichier.php');
echo '<h1>RESULTATS</h1>';
```

- Ce code affiche deux balises h1 avec CODE PHP puis RESULTATS
- La fonction « highlight_file » permet d'afficher le contenu du fichier proposé. Quand on teste le code, on commence par affiche le code. Ca permet de voir le code en même temps que les résultats.
- Pour généraliser le code, on écrit : highlight_file(basename(__FILE__));
- basename(__FILE__) permet de récupérer le nom du fichier en cours de traitement.

Document « slidé »

- Ce document Word est en partie « slidé » : chaque page tient, en général, sur un écran, comme un slide.

1. Méthode POST - Formulaires - Le tableau $_POST

Les formulaires de saisie en HTML

Principes

- Les formulaires sont l'outil de base pour échanger des informations avec les visiteurs.
- Un formulaire permet de passer des paramètres (comme dans un appel de fonction) de page à page.
- Les paramètres se retrouvent dans un tableau associatif de paramètres : $_GET ou $_POST
- Le code HTML de la page appelante remplit le $_GET ou le $_POST
- Dans la page appelée, on accède aux éléments du tableau associatif $_GET ou $_POST

Les 4 propriétés fondamentales d'un formulaire

- action : pour savoir quelle page on appelle
- method : pour savoir dans quelle variable, $_GET ou $_POST, on fait transiter l'information
- name : pour avoir la clé de la variable dans le $_GET ou $_POST.
- value : pour avoir la valeur d'une variable dans le $_GET ou $_POST.

Références

- http://www.w3schools.com/html/html_forms.asp
- http://www.w3schools.com/tags/att_form_action.asp
- http://bliaudet.free.fr/IMG/pdf/HTML_4_Fonctionnalites_avancees.pdf

Syntaxe de base du formulaire

- Exemple :


```
<form action="actionPOST.php" method="POST" >
    <label>nom : </label>
    <input type="text" name="nom" maxlength="10" placeholder="votre nom" >
    <input type="submit" value="Valider">
</form>
```

➔ La balise form contient 2 attributs principaux : method et action.

➔ La balise form contient des balises qui permettent de :

1. caractériser le type de formulaire par le type de la balise <input> ou autre pour les champs de saisie, les boutons, les menu déroulant, etc.
2. fournir des variables à passer à la page appelée via les tableaux associatifs $_GET ou $_POST.

L'attribut method de la balise <form>

- **L'attribut method** permet de choisir la technique de transfert d'informations. Il y en a 2 : GET ou POST.
- La méthode GET est celle déjà vue qui fait transiter l'information dans l'URL. On limite l'information à 256 caractères.
- La méthode POST permet d'envoyer des gros contenus d'informations et de cacher le transit aux utilisateurs.
- On utilise préférentiellement la méthode POST dans les formulaires.

L'attribut action de la balise <form>

- **L'attribut action** donne l'URL de la page qui sera appelée avec cette balise form, quand on valide avec le bouton correspondant à la balise <input> de type submit.
- Ce sera une page PHP (ou n'importe quel autre page d'un langage serveur) qui sera capable de traiter les informations transmises.
- On peut aussi faire appel à la même page.

2 balises associées à la balise form

- **La balise fieldset :** permet de regrouper dans un cadre plusieurs éléments de saisie. La balise legend joue le rôle d'un label pour ce cadre.
- **La balise label :** elle permet de mettre un texte associé à la zone de saisie.

Les balises à la place d'input

- **La balise input :** c'est la principale balise de saisie. Elle fournit en général un champ de saisie. Mais aussi des saisie « typées » : téléphone, mail, couleurs, etc. Elle fournit aussi les boutons de validation.
- **La balise textarea :** fournit une zone de texte à saisir
- **La balise select :** fournit un menu déroulant
- **La balise button :** souvent utiliser avec un type submit.

Les 2 attributs fondamentaux des balises contenues dans <form> : name et value

- **name** :
  - Pour chaque élément d'un formulaire (un champ de saisie, un bouton, un menu déroulant, etc.), on peut définir un attribut « name » qui sera le nom de la variable qu'on retrouvera dans $_GET ou $_POST et qui sera accessible dans la page appelée.
  - L'attribut « name » est donc une variable passée en paramètre pour la page appelée via $_GET ou $_POST.
- **value** :
  - L'attribut « value » sera la valeur de la variable définie par le « name ».
  - Cette valeur peut aussi être la valeur saisie par l'utilisateur.

3 attributs fondamentaux du contrôle d'interface

- **type :** number, date, tel, password, range, color, etc : changent l'interface !
  - http://www.w3schools.com/html/html_form_input_types.asp
- **required :** sans Valeur, vaut true : impose la saisie. Change l'interface.
- **placeholder :** permet de mettre une valeur en grisé dans la zone de saisie

Exemple-4 : formulaire-POST

Page appelante : < form action="actionPOST.php" method="POST" >

- Code du formulaire

```
<form action="actionPOST.php" method="POST" >
    <p>
        <label for="prenom">Prenom</label>
        <input type="text" name="prenom" id="prenom" placeholder="prenom">
    </p>
    <p>
        <label for="nom">nom</label>
        <input type="text" name="nom" id="nom" placeholder="votre nom">
    </p>
    <p><input type="submit" value="Valider"></p>
</form>
```

- Dans l'input :
  - Le type définit le type de saisie : ici du texte
  - Le name est une clé du tableau associatif $_POST ou $_GET, ici $_POST
  - L'id sert pour le CSS
  - Si le for du <label> est égal à l'id, un clic sur le label conduit dans l'input pour éviter les confusions.
  - placeholder : une info pour la saisie, en grisée

Page appelée : $_POST

```
<p> Bonjour
    <?php echo $_POST['prenom'] .' '. $_POST['nom'] ; ?>
</p>
```

- Le tableau $_POST fonctionne comme le tableau $_GET : c'est un tableau associatif.
- Les name sont des clés (key) du tableau $_POST : ce sera le cas pour tous les autres formulaires.
- Quand on teste, on constate qu'il n'y a pas d'information dans l'URL.

Débogage

- On peut afficher le contenu du tableau $_POST au début de la page :

```
echo '<h3>TABLEAU $_POST : </h3>';
echo '<pre>' ; print_r($_POST); echo '</pre>';
```

exemple 5 : formulaire-GET

Principes

- A la place d'un POST, on peut faire un GET dans le formulaire.
- Dans ce cas, les informations transiteront via l'URL.

Page appelante : < form action="actionPOST.php" method="GET" >

- Code du formulaire

```
<form action="actionGET.php" method="GET" >
```

➔ La page appelée s'appellera « actionGET.php »

Page appelée : $_GET

- Dans la page appelée, actionGET.php, on utilise le tableau $_GET.

```
<p> Bonjour
     <?php echo $_GET['prenom'] .' '. $_GET['nom'] ; ?>
</p>
```

➔ Dans ce cas, les variables transiteront via l'URL. On peut les voir dans l'URL.

2. Toujours vérifier les informations fournies par l'utilisateur – 2 : faille XSS

Never trust user input !

Principe : envoyer du code : faille XSS

- Avec la méthode POST, l'information envoyée est cachée.
- Mais on peut envoyer n'importe quoi et particulièrement du code HTML ou JavaScript qui peut nuire à l'utilisation normale du site, en affichant par exemple des contenus inadaptés.
- L'envoi de code à la place d'un texte, c'est ce qu'on appelle la **faille XSS**.

Se protéger contre les failles : aide en ligne

- https://openclassrooms.com/fr/courses/6179306-securisez-vos-applications-web-avec-lowasp/6520368-stoppez-le-cross-site-scripting-xss

Exemple-s 4 et 5

- Les exemples précédents ne sont pas protégés.
- On peut saisir une simple balise HTML, ou du code JavaScript, ou des balises HTML complexes (un formulaire !)

Saisie de balise

- Dans le test précédent, à la place de saisir un simple nom : « Bertrand », on peut saisir
  - ➔ <h1>Bertrand</h1>
- Dans ce cas, la balise <h1> sera interprétée et le texte apparaitra en très gros.

Saisie de code javascript

- On peut aussi ajouter un code JavaScript :
  - ➔ <script type="text/javascript">alert('Un virus a été détecter ! Veuillez vous rendre sur bliaudet.free.fr')</script> !

- Les 2 exemples précédents ne sont pas très graves, mais avec du code JavaScript, un pirate peut récupérer les informations privées d'un utilisateur.

Rôle des navigateurs

Safari (apple) protège automatiquement contre les intrusions XSS.
Pas Firefox ni Chrome.

Solution : exemple-6 : htmlspecialchars() et htmlentities()

Principes

- PHP fournit deux fonctions qui permettent de traiter le code interprétable comme une simple chaîne de caractères.
  - ➔ **htmlspecialchars()** : http://php.net/manual/fr/function.htmlspecialchars.php
  - ➔ **htmlentities()** : http://php.net/manual/fr/function.htmlentities.php

Fonction htmlspecialchars

- La fonction htmlspecialchars permet que le code inséré soit traité comme du texte normal.
  - ➔ htmlspecialchars($_POST['nom'])
- Ainsi si on saisit : <h1>Bertrand</h1>, le site affichera : bonjour <h1>Bertrand</h1> : la balise h1 n'aura pas été prise en compte.

Fonction htmlentities

- La fonction htmlentities est équivalente à la fonction htmlspecialchars en encore plus restrictive : tous les caractères qui ont des équivalents HTML sont traduits.

Principe de résolution : remplacement du caractère spécial « < » en caractère normal « < »

- Quand on regarde le code source d'une page avec un htmlspecialchars, on voit que le « < » qui est interprété par le navigateur comme le début d'une balise est remplacé par « < » qui permet au navigateur d'afficher un <
- C'est le cas dans Safari, mais pas dans Firefox ni Chrome.

Fonctions de traitement de chaînes de caractères

exemple-6

- Htmlspecialchars et htmlentities sont des fonctions de traitement de chaîne de caractères.
- Toutes les fonctions : http://php.net/manual/fr/ref.strings.php
- On y trouve par exemple :
  - ➔ ltrim (suppression des espaces en trop),
  - ➔ ucfirst (upper case pour la première lettre), etc.

printf et sprintf

- printf et sprintf sont 2 autres fonctions de traitement de chaine de caractères.
  - ➔ La fonctions printf permet d'écrire une chaine avec des variables « formatés (comme un « echo » mais formaté).
  - ➔ La fonction sprintf permet de retourner le résultat dans une chaine.

- Exemple printf

```
printf(« Le prix est %.2f euros », $prix) ;
```

  - ➔ Le résultat affiché pour le prix le sera avec 2 chiffres après la virgule.
- Syntaxe
  - %s : string
  - %f : float avec plusieurs chiffres après la virgule.
  - %d : entier
  - %c : caractère
  - etc.
  - Toute la documentation : http://php.net/manual/fr/function.sprintf.php
  - Tester du code : http://phptester.net

- Exemple sprintf

```
$msg = sprintf(« Le prix est %.2f euros », $prix) ;
echo $msg
```

TP - $_POST

3. TP-1 : Exercice password

TP-1 : Version 1

Objectif

- Donner l'accès à une page qui contient des informations cachées pour ceux qui ont le mot de passe. Si on n'a pas le bon mot de passe, on revient sur la page d'accueil.

Méthode

- D'abord réfléchir sur papier !
- Quelles sont les pages en jeu ? Une, plusieurs ?
- Comment les pages communiquent-elles entre elles ?

Première analyse

- Une page d'accueil : un formulaire de saisie d'un mot de passe. En HTML.
- Le mot de passe saisi dans le formulaire est envoyé dans le tableau $_POST à la page action, en PHP, qui vérifie que le mot de passe est bon avant d'afficher les informations cachées.
- Si le mot de passe est faux, il faut afficher un message d'erreur
- Dans tous les cas, la page appelée permet de revenir sur la page appelante.

Elément de solution

- Dans la page action on trouve ce code :

```
if(isset($_POST['password']) AND
htmlspecialchars($_POST['password'])=='password'){
```

Remarque : crypter les mots de passe

Circulation sur le réseau : https

- Le mot de passe apparaît donc dans la page action. Toutefois, il est dans le code php (dans un if), il ne sera donc jamais visible sur navigateur du poste client. C'est donc correctement sécurisé si le serveur est lui aussi sécurisé !

- MAIS : si on fait circuler le mot de passe sur le réseau, il y a un risque. C'est le https qui permet de sécuriser la circulation.

Enregistrement dans la BD

- En pratique, le mot de passe sera récupéré dans la BD :

- MAIS : les mots de passe doivent être enregistrés cryptés dans la BD.
- Pour crypter, on utilise la fonction crypt :

```
$ hashed_password = crypt($password, "st")
```

On passe l'argument "st": regarder la doc pour comprendre.

- Pour décrytpter, on écrit le code suivant, password étant le mot de passe saisi, hashed_password le mot de passe récupéré dans la BD.

```
if (crypt($password, "st") == $hashed_password) {
   echo "Mot de passe correct !";
}
else{
  echo "Mot de passe incorrect !";
}
```

TP-1 : Version 2 : monofichier

- On souhaite tout coder dans un seul fichier.
  ➔ La page action est la page d'accueil elle-même.

- Il va falloir tester les différents cas au début de la page d'accueil :
  ➔ Est-on dans le cas où il faut afficher le formulaire ? $_POST['password'] n'est pas setté
  ➔ Est-on dans le cas où le mot de passe est le bon ? $_POST['password'] = « password »
  ➔ Est-on dans le cas où le mot de passe est mauvais ? $_POST['password'] != « password »

TP-1 : Version 3 : monofichier sur le $_GET + MVC + multi-fichiers

- On souhaite tout coder dans un seul fichier
  ➔ et gérer du MVC propre (normalement c'est déjà fait)
  ➔ et gérer du multi-fichiers : ici on va sortir la vue dans un fichier et faire une vraie page HTML.