php

3


```
 *
 * @var boolean
 */
define('PSI_INTERNAL_XML', false)

if (version_compare("5.2", PHP_V
    die("PHP 5.2 or greater is r

}
if (!extension_loaded("pcre"))
    die("phpSysInfo requires th
        properly.");

}

require_once APP_ROOT.'/includes/autoloader.inc.php';

// Load configuration
require_once APP_ROOT.'/config.php';

if (!defined('PSI_CONFIG_FILE') || !defined('PSI_DEBUG')) {
    $tpl = new Template("/templates/html/error_config.html");
    echo $tpl->fetch();
    die();
                                    t javascript
                                    l . strtolower(
```

Messmer Lucie

Dans ce cours :

- Rappel sur les formulaires
- Méthode GET et méthode POST
- Envoyer des données
- Récupérer des données
- isset() et empty()
- Sécurité

Exercice PHP4

Envoyer des données

Formulaires

Les formulaires sont un outil primordial lors du développement d'un site Web. C'est grace à eux que l'utilisateur va pouvoir transmettre des informations et ainsi réaliser n'importe quelle action pouvant être executée par un script.

Rappel <form>

Tout formulaire HTML commence et se termine par la balise <form> </form>.
Les champs permettant de transmettre les informations sont les balises <input /> et se placent à l'interieur.

Syntaxe

```
<form>
  <label for="id_de_mon_champ">Label de mon input </label>
  <input type="text" name="nom_de_mon_champ">
</form>
```

Les attributs **type** et **name** sont obligatoires si l'on veut transmettre nos données correctement).

Il existe la balise <textarea> </textarea> également pour des champs de formulaires multilignes.

Il existe également des types liés aux inputs permettant l'envoie du formulaire <input type="submit"> et <input type="hidden">

<input type="hidden">

```
<input type='hidden' name='invisible' value='val'>
```

Cet input permet de transmettre une information de notre choix de maniere cachée.

Attention : l'input n'apparait pas pour l'utilisateur mais s'il observe le code de la page avec le devTool, il pourra tout à fait lire la donnée transmise. Cet input ne doit donc pas etre utilisé pour faire passer des informations cruciales nécessitant une sécurité.
On l'utilisera pour transmettre toute donnée ne nécessitant aucun choix de la part de l'utilisateur.

input type : SUBMIT, RESET et BUTTON

```
<input type='submit' value='envoyer'>
```

Cet input fait apparaître un bouton dont le clic envoi le formulaire.

```
<input type='reset' value='effacer'>
```

Cet input fait apparaître un bouton dont le clic réinitialise le formulaire avec ses valeurs par défaut.

```
<input type='button' value='bouton'>
```

Cet input fait apparaître un bouton neutre : un clic sur ce dernier ne génère aucune action. Son intérêt est de lui associer du code (JS) qui nous permettra de lancer une action de notre choix.

Transmettre des données

Maintenant que nous savons construire un formulaire et récupérer des informations auprès de l'utilisateur, voyons comment transmettre ces dernières à un script PHP capable de les traiter.

Destination

C'est dans la balise <form> avec l'attribut action que nous allons préciser où envoyer ces données :

```
<form action='fichier.php'>
```

NB :

- Si le fichier de destination n'est pas préciser, les données seront traiter sur la même page.
- Il est tout à fait possible d'appeller un script sur un autre serveur.

Les méthodes pour récuprérer les données :

Il existe plusieurs méthodes pour récuperer des données d'un formulaire : GET et POST.
Ces deux méthodes fonctionnent d'une façon très similaire.

Syntaxe :

```
<form action="ma_page.php" method="get">
```

Méthode GET

Cette méthode est la méthode dite par défaut.

```
<form action='fichier' method='get'>
```

Les donnees sont transmises 'en clair' dans l'URL, c'est a dire qu'on peut les lire sans problèmes à travers ce qu'on appelle la 'query string'.
Elle commence par un **?** (point d'interrogation) qui suit le nom du fichier de destination, puis suivent les couples **nom=valeur** separés par des **&** (esperluette).

Exemple de conteniu de l'URL : mon_script.php?banane=4&pomme=1&ananas=3

Cette méthode est souvent utilisée pour débuguer ou tester son code, mais rarement pour transmettre des données et ce, principalement pour trois raisons :
- Manque de securite : toutes les informations transmises sont lisibles dans l'URL.
- Limitation de taille : les navigateurs limitent la taille de l'URL a en general entre 256 et 512 caractères.
- URL encodage : certains caractères doivent etre encodés dans l'URL. Cela est fait automatiquement par le formulaire mais si l'on souhaite nous même la générer, les choses se corsent.

exemple :
prenom=Eric devient prenom=%C3%89ric

texte=Bonjour a tous devient texte=Bonjour+%C3%A0+tous

operateur=b&you devient operateur=b%26you

Méthode POST

```
<form action='fichier.php' method='post'>
```

Avec cette méthode, les données sont transmises dans l'en-tête HTTP de manière masquée.

Attention : *masquée ne veut pas dire cryptée (comme avec le protocole HTTPS) ce qui signifit qu'une personne mal intentionnée pourrait intercepter et lire ces informations.*

De plus, cette methode permet d'envoyer une quantité d'informations illimitée, et c'est la seule qui autorise l'envoi de fichiers en precisant dans la balise form un nouvel attribut pour l'encodage : **enctype**

```
<form action='fichier' method='post' enctype='multipart/form-data'>
```

Récupération

La récupération des données en provenance d'un formulaire se fait à l'aide de super variables.

Ces variables vont contenir tout simplement ces données sous forme de tableau.

$_GET Fonctionne comme un tableau associatif dont chaque mot-clef correspond à un nom de champ (name) et auquel est bien evidemment associe la valeur correspondante.

Pour les données passees à l'aide de la méthode POST, on utilisera plutôt **$_POST.**

Il existe également une variante qui contient les deux méthodes : **$_REQUEST** .

Exemple avec $_GET (similaire avec *$_POST ou $_REQUEST)* :

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

<form action='traitement.php' method='get'>
      <input type='text' name='prenom'>
      <input type='text' name='nom'>
      <input type='submit' value='ENVOYER'>
</form>
```

Certaines données peuvent être envoyées sous forme de tableau.

C'est le cas par exemple pour des checkbox associées ou une liste de sélection à choix multiple.

Rien ne change au niveau du PHP, les informations sont récupérées de la même manière.

```
<?php
 $options = $_GET['option'];
 print r($options);
?>
<form action='traitement.php' method='get'>
 <input type='checkbox' name=' option[]' id='option1' value='1'>Rouge
 <input type='checkbox' name='option[]' id='option2' value='2'>Vert
 <input type='checkbox' name='option[]' id='option3' value='3'>Bleu
 <input type='submit' value='ENVOI'>
</form>
```

isset()

Il peut arriver dans certains cas que le script PHP tente de récuperer la valeur d'un **champ qui n'a pas été transmis** (notamment dans le cas d'une checkbox non cochée), ce qui va générer une erreur.
Afin d'éviter cela, il est d'usage de vérifier l'existence d'une variable avant récuperation grâce à la fonction **isset()**.
exemple :

```
var_dump(isset($test)); // va afficher : bool(false)
$test = 'ok';
var_dump(isset($test)); // va afficher : bool(true)
```

```
if (isset ($_GET['accord'])) {
      $accord = 'oui';
 } else {
      $accord = 'non';
}
```

Attention : Les select en mode 'selection multiple', les checkbox et les radio ne renvoient rien si l'utilisateur n'a fait aucun choix ou que les cases sont décochées. Les tests avec isset() deviennent primordiaux !

empty(variable)

Cette fonction alternative permet de vérifier non seulement l'existence d'une variable, comme **isset()**, mais également le fait qu'elle ne soit pas vide. Les vérifications javascript côté client pouvant être désactivées ou modifiées, elles ne suffisent pas à assurer la récupération correcte des données souhaitées. Tous les tests devront être doublés côté serveur, en PHP.
(taille minimum ou maximum d'un texte, type de donnee, plage correcte, validite d'une adresse email, respect d'un format particulier...)

Réception (PHP)

Lorsque le formulaire est soumis, le fichier est téléchargé sur le serveur hébergeant le formulaire, dans un dossier temporaire. Aucune autre action n'est enclanchant, c'est à nous de choisir ce que l'on va faire du fichier.
Nous allons utiliser la super variable **$_FILES** pour récupérer toutes les informations utiles sur les fichiers qui viennent d'être téléchargés :

`$_FILES['monfichier']['tmp_name']`	Le nom et l'emplacement temporaire du fichier
`$_FILES['monfichier']['name']`	Le nom original du fichier
`$_FILES['monfichier']['type']`	Son type MIME (image/gif, application/pdf...)
`$_FILES['monfichier']['size']`	Sa taille en octets (1 000 000 octets = 1Mo pour faire simple)
`$_FILES['monfichier']['error']`	Contient zero si tout s'est bien passé, sinon un code d'erreur.

Tout comme pour les données plus classiques reçues d'un formulaire, il nous faut procéder à toute une batterie de test avant de valider le fichier.

Le fichier existe t-il ?

```
if (isset($_FILES['monfichier'])) {
   // Ici on continu...
}
```

Le téléchargement s'est-il bien passé ? La taille ne depasse t'elle pas la limite fixée ? Le type du fichier est-il celui attendu ?

```
if ($_FILES['monfichier']['error'] == 0)     {
    if ($_FILES['monfichier']['size'] < 2000000)     {
       if ($_FILES['monfichier']['type'] == 'image/png') {
        // Ici on continu...
        } else {
         //error type
        }
    } else {
        // error size
    }
  } else {
   // error recuperation
}
```

move_uploaded_file

Lorsque tout est bon, on va copier le fichier où bon nous semblera grâce a une fonction dédiée :

`move_uploaded_file(chemin_temporaire, chemin_final)`

Lorsqu'on défini le chemin final pour notre fichier, nous pouvons le renommer.

exemple :

```
move_uploaded_file($_FILES['monfichier']['tmp_name'],'download/fichier.txt');
move_uploaded_file($_FILES['monfichier']['tmp_name'],'download/'.$_FILES['monfichier'] ['name']);
```

Sécurité

Les formulaires font partie des points les plus vulnerables des applications web. Si certaines failles sont facilement évitées, d'autres peuvent poser plus de problèmes et demander une attention particulière. Les informations que nous collectons provenant de l'exterieur, nous devons absolument les verifier avant de les valider.

Défenses au niveau du HTML

L'utilisateur pouvant voir le code et même le copier pour l'utiliser depuis une de ses pages, tout ce que nous essaierons de faire dans notre formulaire HTML ne peut être considéré comme sûr.
Utiliser l'attribut max_file_size pour limiter la taille d'un fichier à envoyer peut être contourné, tout comme un input de type hidden peut être lu.

La seule chose à eviter est l'utilisation de la méthode get si on veut garder un minimum de confidentialité pour nos données. Mais là encore, des logiciels permettant la lecture des en-têtes HTTP d'une requête en post sont malheureusement à la portée de n'importe qui aujourd'hui.

Défenses au niveau du script

Comme pour le HTML, les scripts étant côté client, ils pourront facilement être contournés ou supprimés. Ils doivent être présent pour le confort de l'utilisateur et pour pallier aux erreurs ou tentatives malveillantes.

Défenses au niveau du PHP

Ce n'est veritablement qu'ici que nous allons pouvoir agir efficacement contre les tentatives mal intention-nees et les erreurs.

exemple :

```
if (isset($_POST['prenom'])) { //
Cette variable existe } else {
  // Cette variable n'existe pas
}
if ($_POST['prenom']!='')    {
  // Cette variable n'est pas vide }
else {
  // Cette variable est vide
  }
```

Tester le type

Tester le type va etre tres important surtout si l'on veut stocker ces informations dans une base de donnee. On a des equivalents pour tous les types :

- is_int() entier
- is_float() nombre decimal
- is_numeric entier ou decimal
- is_bool() booleen tableau
- is_array()

```
if (is_string($_POST['prenom'])) {
  // Cette variable est bien une chaine de caractere }
else {
  // Cette variable n'est pas une chaine de caractere
}
```

Tester la taille

Tester la taille d'une variable permet de refuser une certaine taille de données par exemple.

```
if (mb_strlen($_POST['prenom'])>25) {
  // Cette variable est trop longue } else {
  // Cette variable a une taille correcte
}
if ((isset($_POST['prenom']) && ($_POST['prenom']!='')) {
  // On récupère la valeur passée } else {
  // On attribue une valeur par défaut, par exemple 'inconnu'
}
```

Tester avec des REGEXP ou filter_var

Les expressions régulières sont parfaites pour vérifier les informations qui doivent respecter un format bien précis.

```
$regexp = "/^[0-9][A-z0-9_]+([.][A-z0-9_]+)*[@][A-z0-9_]+([.][A-z0-9_]+)*[.][A-z] {2,4}$/";
if (preg_match($regexp, $_POST['email']) {
   // Email valide }
 else {
   // Email non valide
 }
 if (filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {
 // Email valide
 } else {
 // Email non valide
 }
```

Contrer les injections

Dans un champs texte par exemple, au lieu de simplement placer une chaine de caractère inoffensive, nous allons placer le code suivant :

```
'><script>alert('BOUM !')</script>
```

Ou celui-ci :

```
'><a href='http://www.google.fr'>mon site</a><img src='pirate.gif' title='ah ah ah
```

En utilisant un code HTML/PHP classique nous constatons rapidement qu'il y a un probleme :

```
<input type='text' name='prenom' value='<?php echo $prenom; ?>'>
```

En effet, tous les caractères ont été insérés et certains sont interprétés par le HTML, **changeant par la même occasion le code d'origine.** Pour éviter cela, nous devons transformer tous les caractères speciaux HTML des informations que nous recevons.

htmlspecialchars(str, options)

Cette fonction renvoi la string passée en encodant en HTML tout ce qui peux poser problème. Parmis les nombreuses options, deux nous intérèssent particulièrement :

ENT_COMPAT convertit les guillemets doubles, ignore les simples

ENT_QUOTES convertit les guillemets doubles et simples

```
$test = "'><script>alert('BOUM !')</script>";
echo htmlspecialchars($test, ENT_QUOTES);
// Va afficher : &#039;&gt;&lt;script&gt;alert(&#039;BOUM !&#039;)&lt;/script&gt;
```

De cette façon, l'envoie de code pour attaquer le site, sera empêché.