

## Chapitre 4 : Transmettre des données de page en page

Maintenant que vous avez acquis les bases de PHP, nous pouvons nous intéresser à du concret.

Le langage PHP a été conçu pour que vous puissiez transmettre des informations de page en page, au fil de la navigation d'un visiteur sur votre site. C'est notamment ce qui vous permet de retenir son pseudonyme tout au long de sa visite, mais aussi de récupérer et traiter les informations qu'il rentre sur votre site, notamment dans des formulaires.

Grâce à cette partie, vous allez pouvoir commencer à communiquer vraiment avec vos visiteurs !

### Transmettre des données avec l'URL

Savez-vous ce qu'est une URL ? Cela signifie **Uniform Resource Locator**, et cela sert à représenter une adresse sur le web. Toutes les adresses que vous voyez en haut de votre navigateur, comme <http://www.ofppt.com>, sont des URL.

Je me doute bien que vous ne passez pas votre temps à les regarder (il y a bien mieux à faire !), mais je suis sûr que vous avez déjà été surpris de voir que certaines URL sont assez longues et comportent parfois des caractères un peu curieux. Par exemple, après avoir fait une recherche sur Google, la barre d'adresse contient une URL longue qui ressemble à ceci :

<http://www.google.fr/search?rlz=1C1GFR343&q=ofppt>

Dans cet exemple, les informations après le point d'interrogation sont des données que l'on fait transiter d'une page à une autre. Nous allons découvrir dans ce chapitre comment cela fonctionne.

### Envoyer des paramètres dans l'URL

En introduction, je vous disais que l'URL permettait de transmettre des informations. Comment est-ce que ça fonctionne exactement ?

### Former une URL pour envoyer des paramètres

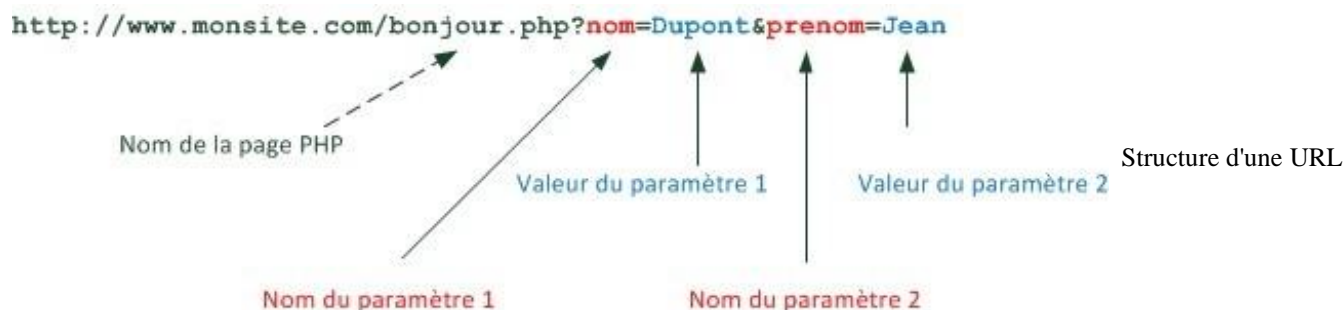
Imaginons que votre site s'appelle `monsite.com` et que vous avez une page PHP intitulée `bonjour.php`. Pour accéder à cette page, vous devez aller à l'URL suivante :

`http://www.monsite.com/bonjour.php`

Jusque-là, rien de bien nouveau. Ce que je vous propose d'apprendre à faire, c'est d'**envoyer** des informations à la page `bonjour.php`. Pour cela, on va ajouter des informations à la fin de l'URL, comme ceci :

`http://www.monsite.com/bonjour.php?nom=Dupont&prenom=Jean`

Ce que vous voyez après le point d'interrogation, ce sont des **paramètres** que l'on envoie à la page PHP. Celle-ci peut récupérer ces informations dans des variables. Voyez sur la figure suivante comment on peut découper cette URL.



Le point d'interrogation sépare le nom de la page PHP des paramètres. Ensuite, ces derniers s'enchaînent selon la forme `nom=valeur` et sont séparés les uns des autres par le symbole `&`.



On peut écrire autant de paramètres que l'on veut ?

En théorie, oui. Il suffit de les séparer par des `&` comme je l'ai fait. On peut donc voir une URL de la forme :

page.php?param1=valeur1&param2=valeur2&param3=valeur3&param4=valeur4...

La seule limite est la longueur de l'URL. En général il n'est pas conseillé de dépasser les 256 caractères, mais les navigateurs arrivent parfois à gérer des URL plus longues. Quoi qu'il en soit, vous aurez compris qu'on ne peut pas non plus écrire un roman dans l'URL.

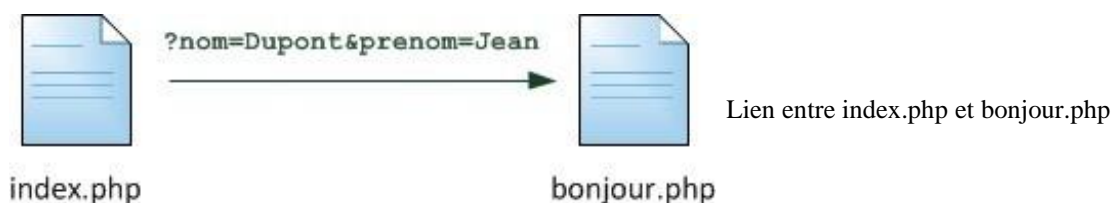
## Créer un lien avec des paramètres

Maintenant que nous savons cela, nous pouvons créer des liens en HTML qui transmettent des paramètres d'une page vers une autre.

Imaginons que vous avez deux fichiers sur votre site :

- index.php (l'accueil) ;
- bonjour.php.

Nous voulons faire un lien de index.php qui mène à bonjour.php et qui lui transmet des informations dans l'URL, comme le schématise la figure suivante.



Pour cela, ouvrez index.php (puisque c'est lui qui contiendra le lien) et insérez-y par exemple le code suivant :

### Code : PHP

```
<a href="bonjour.php?nom=Dupont&prenom=Jean">Dis-moi bonjour  
!</a>
```



Comme vous le voyez, le & dans le code a été remplacé par &amp; dans le lien. Ça n'a rien à voir avec PHP : simplement, en HTML, on demande à ce que les & soient écrits &amp; dans le code source. Si vous ne le faites pas, le code ne passera pas la validation W3C.

Ce lien appelle la page bonjour.php et lui envoie deux paramètres :

- nom : Dupont ;
- prenom : Jean.

Vous avez sûrement deviné ce qu'on essaie de faire ici : on appelle une page bonjour.php qui va dire « Bonjour » à la personne dont le nom et le prénom ont été envoyés en paramètres.

Comment faire dans la page bonjour.php pour récupérer ces informations ? C'est ce que nous allons voir maintenant. ;-)

## Récupérer les paramètres en PHP

Nous savons maintenant comment former des liens pour envoyer des paramètres vers une autre page. Nous avons pour cela ajouté des paramètres à la fin de l'URL.

Intéressons-nous maintenant à la page qui réceptionne les paramètres. Dans notre exemple, il s'agit de la page bonjour.php. Celle-ci va automatiquement créer un array au nom un peu spécial : \$\_GET. Il s'agit d'un array associatif (vous vous souvenez ? Nous avons étudié cela il y a peu !) dont les clés correspondent aux noms des paramètres envoyés en URL.

Reprenons notre exemple pour mieux voir comment cela fonctionne. Nous avons fait un lien vers bonjour.php?nom=Dupont&prenom=Jean, cela signifie que nous aurons accès aux variables suivantes :

Nom	Valeur
<code>\$_GET['nom']</code>	Dupont
<code>\$_GET['prenom']</code>	Jean

On peut donc récupérer ces informations, les traiter, les afficher, bref faire ce que l'on veut avec. Pour commencer, essayons tout simplement de les afficher.

Créez un nouveau fichier PHP, appelez-le `bonjour.php` et placez-y le code suivant



Bien sûr, pour une vraie page web il faudrait écrire toutes les informations d'en-tête nécessaires en HTML : le doctype, la balise `<head>`, etc. Ici, pour faire simple, je ne mets pas tout ce code. La page ne sera pas très belle (ni valide W3C, d'ailleurs) mais ce n'est pas encore notre problème : pour l'instant, nous faisons juste des tests.

#### Code : PHP

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

Ici, nous affichons le prénom qui a été passé dans l'URL. Bien entendu, nous avons aussi accès au nom. Nous pouvons afficher le nom et le prénom sans problème :

#### Code : PHP

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

Comme vous le voyez, j'en ai profité pour faire une petite concaténation, comme nous avons appris à le faire. Ce code que vous voyez là n'est pas bien complexe : nous affichons le contenu de deux cases de l'array `$_GET`. C'est vraiment très simple et il n'y a rien de nouveau. Tout ce qu'il faut savoir, c'est qu'on peut retrouver les paramètres envoyés dans l'URL grâce à un array nommé `$_GET`.

Vous devriez aussi faire le test chez vous pour vous assurer que vous comprenez bien le fonctionnement ! Si besoin est, vous pouvez télécharger mes fichiers d'exemple `index.php` et `bonjour.php` :

## Ne faites jamais confiance aux données reçues !

Il est impossible de terminer ce chapitre sans que je vous mette en garde contre les **dangers** qui guettent les apprentis webmasters que vous êtes. Nous allons en effet découvrir qu'il ne faut JAMAIS faire confiance aux variables qui transitent de page en page, comme `$_GET` que nous étudions ici.

Le but de cette section est, je l'avoue, de vous faire peur, car trop peu de développeurs PHP ont conscience des dangers et des failles de sécurité qu'ils peuvent rencontrer, ce qui risque pourtant de mettre en péril leur site web ! Oui je suis alarmiste, oui je veux que vous ayez — un peu — peur ! Mais rassurez-vous : je vais vous expliquer tout ce qu'il faut savoir pour que ça se passe bien et que vous ne risquiez rien. ;-)

Ce qui compte, c'est que vous ayez conscience très tôt (dès maintenant) des problèmes que vous pourrez rencontrer lorsque vous recevrez des paramètres de l'utilisateur.

## Tous les visiteurs peuvent trafiquer les URL

Si vous faites les tests des codes précédents chez vous, vous devriez tomber sur une URL de la forme :

`http://localhost/tests/bonjour.php?nom=Dupont&prenom=Jean`

On vous dit bien « Bonjour Jean Dupont ! ». Mais si vous êtes un peu bricoleurs, vous pouvez vous amuser à changer les paramètres directement dans la barre d'adresse, comme dans la figure suivante.



On peut

N'importe qui peut modifier les paramètres  
dans la barre d'adresse de son navigateur !

facilement modifier les paramètres dans le navigateur

Essayez par exemple de modifier l'adresse pour :

`http://localhost/tests/bonjour.php?nom=Dupont&prenom=Marc`

Comme vous le voyez, ça marche ! N'importe qui peut facilement modifier les URL et y mettre ce qu'il veut : il suffit simplement de changer l'URL dans la barre d'adresse de votre navigateur.



Et alors, quel est le problème ? C'est pas plus mal, si le visiteur veut adapter l'URL pour qu'on lui dise bonjour avec son vrai nom et son vrai prénom ?

Peut-être, mais cela montre une chose : **on ne peut pas avoir confiance dans les données qu'on reçoit**. N'importe quel visiteur peut les changer. Dans la page `index.php` j'ai fait un lien qui dit bonjour à Jean Dupont, mais la page `bonjour.php` ne va pas forcément afficher « Bonjour Jean Dupont ! » puisque n'importe quel visiteur peut s'amuser à modifier l'URL.

Je vous propose de faire des modifications encore plus vicieuses et de voir ce qu'on peut faire pour éviter les problèmes qui en découlent.

## Tester la présence d'un paramètre

Allons plus loin. Qu'est-ce qui empêche le visiteur de supprimer tous les paramètres de l'URL ? Par exemple, il peut très bien tenter d'accéder à :

`http://localhost/tests/bonjour.php`

Que va afficher la page `bonjour.php` ? Faites le test ! Elle va afficher quelque chose comme :

### Code : Console

```
Bonjour
Notice: Undefined index: prenom in C:\wamp\www\tests\bonjour.php on line 9

Notice: Undefined index: nom in C:\wamp\www\tests\bonjour.php on line 9
!
```

Que s'est-il passé ? On a essayé d'afficher la valeur de `$_GET['prenom']` et celle de `$_GET['nom']`... Mais comme on vient de les supprimer de l'URL, ces variables n'ont pas été créées et donc elles n'existent pas ! PHP nous avertit qu'on essaie d'utiliser des variables qui n'existent pas, d'où les « Undefined index ».

Pour résoudre ce problème, on peut faire appel à une fonction un peu spéciale : `isset()`. Cette fonction teste si une variable existe. Nous allons nous en servir pour afficher un message spécifique si le nom ou le prénom sont absents.

#### Code : PHP

```
<?php
if (isset($_GET['prenom']) AND isset($_GET['nom'])) // On a le nom
et le prénom
{
    echo 'Bonjour ' . $_GET['prenom'] . ' ' . $_GET['nom'] . ' !';
}
else // Il manque des paramètres, on avertit le visiteur
{
    echo 'Il faut renseigner un nom et un prénom !';
}
?>
```

Que fait ce code ? Il teste si les variables `$_GET['prenom']` et `$_GET['nom']` existent. Si elles existent, on dit bonjour au visiteur. S'il nous manque une des variables (ou les deux), on affiche un message d'erreur : « Il faut renseigner un nom et un prénom ! ».

Essayez maintenant d'accéder à la page `bonjour.php` sans les paramètres, vous allez voir qu'on gère bien le cas où le visiteur aurait retiré les paramètres de l'URL.



Est-ce que c'est vraiment la peine de gérer ce cas ? C'est vrai quoi, moi je ne m'amuse jamais à modifier mes URL et mes visiteurs non plus, je pense !

Oui, oui, trois fois oui : il faut que vous pensiez à gérer le cas où des paramètres que vous attendiez viendraient à manquer.

Vous vous souvenez de ce que je vous ai dit ? Il ne faut jamais faire confiance à l'utilisateur. Tôt ou tard vous tomberez sur un utilisateur malintentionné qui essaiera de trafiquer l'URL pour mettre n'importe quoi dans les paramètres. Il faut que votre site soit prêt à gérer le cas.

Dans notre exemple, si on ne gérait pas le cas, ça ne faisait rien de bien grave (ça affichait juste des messages d'erreur). Mais lorsque votre site web deviendra plus complexe, cela pourrait avoir des conséquences plus ennuyeuses.

## Contrôler la valeur des paramètres

Allons plus loin dans notre tripatouillage vicieux de l'URL. On va modifier notre code source pour gérer un nouveau paramètre : le nombre de fois que le message doit être répété. Les paramètres vont donc être :

`bonjour.php?nom=Dupont&prenom=Jean&repeter=8`

Le paramètre `repeter` définit le nombre de fois que l'on dit « bonjour » sur la page. Sauriez-vous adapter notre code pour qu'il dise bonjour le nombre de fois indiqué ? Voici la solution que je vous propose :

#### Code : PHP

```
<?php
if (isset($_GET['prenom']) AND isset($_GET['nom']) AND
isset($_GET['repeter']))
{
```

```

    for ($i = 0 ; $i < $_GET['repetier'] ; $i++)
    {
        echo 'Bonjour ' . $_GET['prenom'] . ' ' . $_GET['nom'] . ' !<br
    />';
    }
}
else
{
    echo 'Il faut renseigner un nom, un prénom et un nombre de
répétitions !';
}
?>

```

On teste si le paramètre `repetier` existe lui aussi (avec `isset($_GET['repetier'])`). Si tous les paramètres sont bien là, on fait une boucle (j'ai choisi de faire un `for`, mais on aurait aussi pu faire un `while`). La boucle incrémente une petite variable `$i` pour répéter le message de bienvenue le nombre de fois indiqué.

Le résultat ? Si on va sur...

`http://localhost/tests/bonjour.php?nom=Dupont&prenom=Jean&repetier=8`

... alors le message de bienvenue s'affichera huit fois :

#### Code : Console

```

Bonjour Jean Dupont !
Bonjour Jean Dupont !
Bonjour Jean Dupont !
Bonjour Jean Dupont !
Bonjour Jean Dupont !
Bonjour Jean Dupont !
Bonjour Jean Dupont !
Bonjour Jean Dupont !

```

Nous avons amélioré notre page pour qu'elle puisse dire « bonjour » plusieurs fois, et ce nombre de fois est personnalisable dans l'URL. Très bien.

Maintenant, mettez-vous dans la peau du **méchant**. Soyez méchants. Soyez vicieux. Que pourrions-nous faire et que nous n'aurions pas prévu, juste en modifiant l'URL ?

J'ai une idée ! On peut mettre un nombre de répétitions très grand, par exemple 1984986545665156. La page PHP va boucler un très grand nombre de fois et votre PC ne pourra probablement pas supporter une telle charge de travail.

`bonjour.php?nom=Dupont&prenom=Jean&repetier=1984986545665156`

Si vous avez peur d'essayer, je vous rassure : votre PC ne va pas prendre feu en faisant ça. Par contre, il va se mettre à consommer énormément de mémoire pour stocker tous les messages « bonjour » et n'arrivera sûrement pas jusqu'au bout (PHP s'arrête au bout d'un certain temps d'exécution). Ou alors s'il y arrive, votre page va être extrêmement surchargée de messages « bonjour ».



Mon dieu, c'est horrible ! Comment peut-on empêcher ça ?

En étant plus malins et surtout plus prévoyants. Voici ce qu'il faut anticiper.

- Le cas où le nombre qu'on vous envoie **n'est pas une valeur raisonnable**.
  - Par exemple on peut dire que si on dépasse 100 fois, c'est trop, et il faut refuser d'exécuter la page.
  - De même, que se passe-t-il si je demande de répéter « \$-4\$ fois » ? Ça ne devrait pas être autorisé. Il faut que le nombre soit au moins égal à 1.
- Le cas où **la valeur n'est pas logique**, où elle n'est pas du tout ce qu'on attendait. Rien n'empêche le visiteur de remplacer

la valeur du paramètre `repetet` par du texte !

- Que se passe-t-il si on essaie d'accéder à

`bonjour.php?nom=Dupont&prenom=Jean&repetet=grenouille` ?

Cela ne devrait pas être autorisé. Le mot « grenouille » n'a pas de sens, on veut un nombre entier positif.

- Que se passe-t-il si on essaie d'accéder à

`bonjour.php?nom=Dupont&prenom=Jean&repetet=true` ?

Cela ne devrait pas être autorisé non plus, on attendait un nombre entier positif, pas un booléen.

Pour résoudre ces problèmes, on doit :

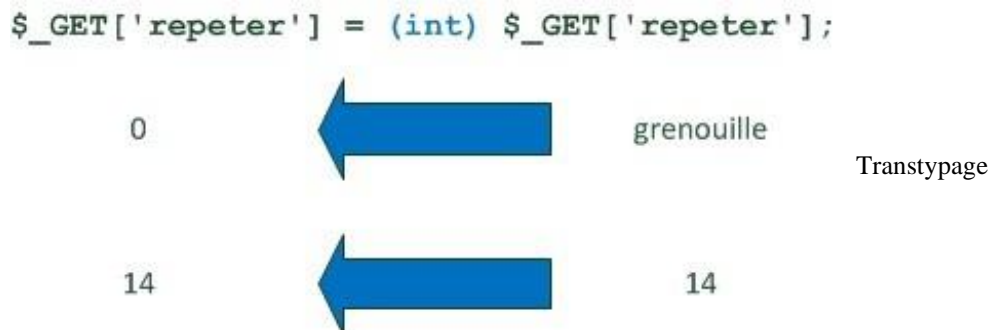
1. vérifier que `repetet` contient bien un nombre ;
2. vérifier ensuite que ce nombre est compris dans un intervalle raisonnable (entre 1 et 100, par exemple).

Pour vérifier que `repetet` contient bien un nombre, une bonne solution pourrait être de forcer la conversion de la variable en type entier (`int`). On peut utiliser pour cela ce qu'on appelle le **transtypage**, une notion qu'on n'a pas encore vue jusqu'ici mais qui est très simple à comprendre. Regardez ce code :

**Code : PHP**

```
<?php
$_GET['repetet'] = (int) $_GET['repetet'];
?>
```

Il faut lire cette instruction de droite à gauche. Le `(int)` entre parenthèses est comme un tuyau de conversion. Tout ce qui transite à travers ressort forcément en une valeur de type `int` (entier). Voyez la figure suivante pour vous en convaincre.



Si la valeur est bien un entier (par exemple 14) alors elle n'est pas modifiée. En revanche, si la variable contient autre chose qu'un entier (par exemple « grenouille »), elle est transformée en entier. Ici, comme PHP ne peut pas associer de valeur à *grenouille*, il lui donne 0.

Après avoir exécuté cette instruction, la variable `$_GET['repetet']` contient maintenant forcément un nombre entier. Il ne reste plus qu'à vérifier que ce nombre est bien compris entre 1 et 100 à l'aide d'une condition (ce que vous savez faire, normalement !).

Voici donc le code final sécurisé qui prévoit tous les cas pour éviter d'être pris au dépourvu par un utilisateur malintentionné :

**Code : PHP**

```
<?php
if (isset($_GET['prenom']) AND isset($_GET['nom']) AND
    isset($_GET['repetet']))
{
    // 1 : On force la conversion en nombre entier
    $_GET['repetet'] = (int) $_GET['repetet'];

    // 2 : Le nombre doit être compris entre 1 et 100
```

```

if ($_GET['repeter'] >= 1 AND $_GET['repeter'] <= 100)
{
    for ($i = 0 ; $i < $_GET['repeter'] ; $i++)
    {
        echo 'Bonjour ' . $_GET['prenom'] . ' ' . $_GET['nom'] . ' !<br
    />';
    }
}
else
{
    echo 'Il faut renseigner un nom, un prénom et un nombre de
répétitions !';
}
?>

```

Cela fait beaucoup de conditions pour quelque chose qui était à la base assez simple, mais c'était nécessaire. Vous devrez toujours faire très attention et prévoir tous les cas les plus tordus, comme nous venons de le faire :

- vérifier que tous les paramètres que vous attendiez sont bien là ;
- vérifier qu'ils contiennent bien des valeurs correctes comprises dans des intervalles raisonnables.

Si vous gardez cela en tête, vous n'aurez pas de problèmes. :-)



Nous n'avons pas encore vu tous les risques liés aux données envoyées par l'utilisateur. Cette première approche devrait déjà vous avoir sensibilisés au problème, mais nous irons plus loin dans le chapitre suivant pour finir de couvrir ce sujet. Tout ceci est un peu complexe, je suis d'accord, mais c'est réellement important !

## En résumé

- Une URL représente l'adresse d'une page web (commençant généralement par `http://`).
- Lorsqu'on fait un lien vers une page, il est possible d'ajouter des paramètres sous la forme `bonjour.php?nom=Dupont&prenom=Jean` qui seront transmis à la page.
- La page `bonjour.php` dans l'exemple précédent recevra ces paramètres dans un array nommé `$_GET` :
  - `$_GET['nom']` aura pour valeur « Dupont » ;
  - `$_GET['prenom']` aura pour valeur « Jean ».
- Cette technique est très pratique pour transmettre des valeurs à une page, mais il faut prendre garde au fait que le visiteur peut les modifier très facilement. Il ne faut donc pas faire aveuglément confiance à ces informations, et tester prudemment leur valeur avant de les utiliser.
- La fonction `isset()` permet de vérifier si une variable est définie ou non.
- Le transtypage est une technique qui permet de convertir une variable dans le type de données souhaité. Cela permet de s'assurer par exemple qu'une variable est bien un `int` (nombre entier).





# Transmettre des données avec les formulaires

Les formulaires constituent le principal moyen pour vos visiteurs d'entrer des informations sur votre site. Les formulaires permettent de créer une **interactivité**.

Par exemple, sur un forum on doit insérer du texte puis cliquer sur un bouton pour envoyer son message. Sur un livre d'or, sur un mini-chat, on procède de la même façon. On a besoin des formulaires partout pour échanger des informations avec nos visiteurs.

Vous allez voir qu'il y a de nombreux rappels de HTML dans ce chapitre... et ce n'est pas un hasard : ici, le PHP et le HTML sont intimement liés. Le HTML permet de créer le formulaire, tandis que le PHP permet de traiter les informations que le visiteur a entrées dans le formulaire.

Ce chapitre est particulièrement important, nous réutiliserons ce que nous avons appris ici dans toute la suite du cours. Soyez attentifs !

## Créer la base du formulaire

En HTML, pour insérer un formulaire, on se sert de la balise `<form>`. On l'utilise de la manière suivante :

### Code : PHP

```
<form method="post" action="cible.php">

<p>
    On insèrera ici les éléments de notre formulaire.
</p>

</form>
```



On écrira le contenu de notre formulaire entre les balises `<form>` et `</form>`. Si vous avez lu mon cours de HTML/CSS, vous verrez qu'une bonne partie de ce chapitre ne sera composée que de rappels puisque je vais vous expliquer comment on insère les champs du formulaire dans la page. Cependant, et ce sera nouveau, nous allons aussi découvrir comment traiter en PHP les données qui ont été envoyées par le visiteur.

Je souhaite attirer votre attention sur la toute première ligne de ce code. Il y a deux attributs très importants à connaître pour la balise `<form>` : la méthode (`method`) et la cible (`action`). Il est impératif que vous compreniez à quoi ils servent.

## La méthode

Il faut savoir qu'il existe plusieurs moyens d'envoyer les données du formulaire (plusieurs « méthodes »). Vous pouvez en employer deux.

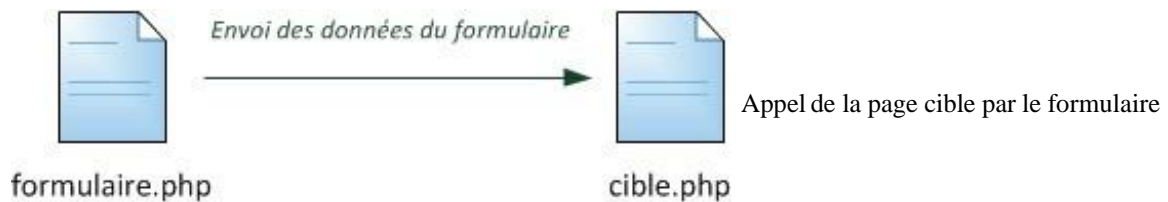
- `get` : les données transiteront par l'URL comme on l'a appris précédemment. On pourra les récupérer grâce à l'array `$_GET`. Cette méthode est assez peu utilisée car on ne peut pas envoyer beaucoup d'informations dans l'URL (je vous disais dans le chapitre précédent qu'il était préférable de ne pas dépasser 256 caractères).
- `post` : les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas passer dans la barre d'adresse. Cette méthode permet d'envoyer autant de données que l'on veut, ce qui fait qu'on la privilégie le plus souvent. Néanmoins, les données ne sont pas plus sécurisées qu'avec la méthode `GET` et il faudra toujours vérifier si tous les paramètres sont bien présents et valides, comme on l'a fait dans le chapitre précédent. **On ne doit pas plus faire confiance aux formulaires qu'aux URL.**

C'est à vous de choisir par quelle méthode vous souhaitez que les données du formulaire soient envoyées. Si vous hésitez, sachez que dans 99 % des cas la méthode que l'on utilise est `post`, vous écrirez donc `method="post"` comme je l'ai fait.

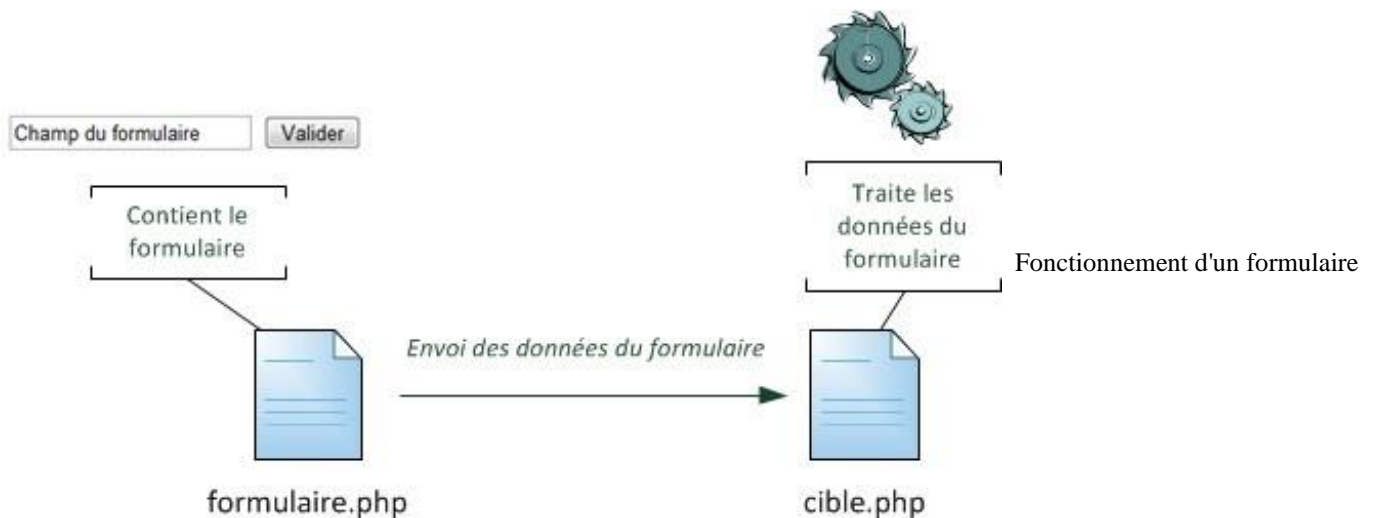
## La cible

L'attribut `action` sert à définir la page appelée par le formulaire. C'est cette page qui recevra les données du formulaire et qui sera chargée de les traiter.

Imaginons le schéma de la figure suivante.



Dans cet exemple, le formulaire se trouve dans la page `formulaire.php`. Cette page ne fait aucun traitement particulier, mais une fois le formulaire envoyé (lorsqu'on a cliqué sur le bouton « Valider »), le visiteur est redirigé vers la page `cible.php` qui reçoit les données du formulaire, comme vous le montre la figure suivante.



Le nom de la page cible est défini grâce à l'attribut `action`.



La page cible ne doit pas forcément s'appeler `cible.php`. J'utilise ce nom dans mes exemples simplement pour que vous compreniez bien que c'est la page qui est appelée. Remarquez qu'en théorie rien n'empêche le formulaire de s'appeler lui-même. Il suffirait d'écrire `action="formulaire.php"`. Dans ce cas, la page du formulaire doit être capable aussi bien d'afficher le formulaire que de traiter les données. C'est tout à fait possible de le faire mais afin de ne pas compliquer les choses trop vite, on va éviter de faire comme ça ici. ;-)

Retenez donc bien que vous travaillez normalement sur deux pages différentes : la page qui contient le formulaire (`formulaire.php` dans notre exemple), et celle qui reçoit les données du formulaire pour les traiter (`cible.php`).

## Les éléments du formulaire

Dans un formulaire, vous le savez peut-être déjà, on peut insérer beaucoup d'éléments différents : zones de texte, boutons, cases à cocher, etc.

Je vais ici tous les énumérer et vous montrer comment vous servir de chacun d'eux dans la page `cible.php` qui fera le traitement. Vous allez voir, c'est vraiment très simple : au lieu de recevoir un array `$_GET`, vous allez recevoir un array `$_POST` contenant les données du formulaire !

## Les petites zones de texte

Une zone de texte ressemble à la figure suivante.

Votre pseudo :  Zone de texte

En HTML, on l'insère tout simplement avec la balise :

**Code : PHP**

```
<input type="text" />
```



Pour les mots de passe, vous pouvez utiliser `type="password"`, ce qui aura pour effet de cacher le texte entré par le visiteur. À part ce détail, le fonctionnement reste le même.

Il y a deux attributs à connaître que l'on peut ajouter à cette balise.

- `name` (obligatoire) : c'est le nom de la zone de texte. Choisissez-le bien, car c'est lui qui va produire une variable. Par exemple :  
`<input type="text" name="pseudo" />`.
- `value` (facultatif) : c'est ce que contient la zone de texte au départ. Par défaut, la zone de texte est vide mais il peut être pratique de pré-remplir le champ. Exemple :  
`<input type="text" name="pseudo" value="M@teo21" />`.

Oui, je sais que vous commencez à vous inquiéter car vous n'avez pas encore vu de PHP pour le moment mais n'ayez pas peur. Le fonctionnement est tout simple et a un air de déjà vu. Le texte que le visiteur aura entré sera disponible dans `cible.php` sous la forme d'une variable appelée `$_POST['pseudo']`.

Pour l'exemple, je vous propose de créer un formulaire qui demande le prénom du visiteur puis qui l'affiche fièrement sur la page `cible.php`. On va donc distinguer deux codes source : celui de la page du formulaire et celui de la page cible.

Voici le code de la page `formulaire.php` :

#### Code : PHP

```
<p>
    Cette page ne contient que du HTML.<br />
    Veuillez taper votre prénom :
</p>

<form action="cible.php" method="post">
<p>
    <input type="text" name="prenom" />
    <input type="submit" value="Valider" />
</p>
</form>
```



Rappel de HTML : le champ `<input type="submit" />` permet de créer le bouton de validation du formulaire qui commande l'envoi des données, et donc la redirection du visiteur vers la page cible.

Maintenant, je vous propose de créer la page `cible.php`. Cette page va recevoir le prénom dans une variable nommée `$_POST['prenom']`.

#### Code : PHP

```
<p>Bonjour !</p>

<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <?php echo
$_POST['prenom']; ?> !</p>

<p>Si tu veux changer de prénom, <a href="formulaire.php">clique
ici</a> pour revenir à la page formulaire.php.</p>
```

Le code web suivant ouvre la page `formulaire.php` pour que vous puissiez tester.

---

[Essayer !](#)

Dans `cible.php` on a affiché une variable `$_POST['prenom']` qui contient ce que l'utilisateur a entré dans le formulaire.

## Les grandes zones de texte

La grande zone de texte (on l'appelle aussi « zone de saisie multiligne ») ressemble à la figure suivante.

Comment pensez-vous que je pourrais améliorer mon site ?

Difficile d'améliorer la perfection non ?

Enfin, moi ce que j'en dis...

A toi de voir !

Une grande zone de texte

On peut y écrire autant de lignes que l'on veut. C'est plus adapté si le visiteur doit écrire un long message, par exemple.

On va utiliser le code HTML suivant pour insérer cette zone de texte :

### Code : PHP

```
<textarea name="message" rows="8" cols="45">
Votre message ici.
</textarea>
```

Là encore, on a un attribut `name` qui va définir le nom de la variable qui sera créée dans `cible.php`. Dans notre cas, ce sera la variable `$_POST['message']`.

Vous remarquerez qu'il n'y a pas d'attribut `value`. En fait, le texte par défaut est ici écrit entre le `<textarea>` et le `</textarea>`. Si vous ne voulez rien mettre par défaut, alors n'écrivez rien entre `<textarea>` et `</textarea>`.



Les attributs `rows` et `cols` permettent de définir la taille de la zone de texte en hauteur et en largeur respectivement.

## La liste déroulante

La figure suivante est une liste déroulante.

Dans quel pays habitez-vous ?



Une liste déroulante

On utilise le code HTML suivant pour construire une liste déroulante :

Code : PHP

```
<select name="choix">
  <option value="choix1">Choix 1</option>
  <option value="choix2">Choix 2</option>
  <option value="choix3">Choix 3</option>
  <option value="choix4">Choix 4</option>
</select>
```

Tout bêtement, on utilise la balise `<select>` à laquelle on donne un nom (ici : « choix »). On écrit ensuite les différentes options disponibles... puis on referme la balise avec `</select>`.

Ici, une variable `$_POST['choix']` sera créée, et elle contiendra le choix qu'a fait l'utilisateur. S'il a choisi « Choix 3 », la variable `$_POST['choix']` sera égale au value correspondant, c'est-à-dire `choix3`.

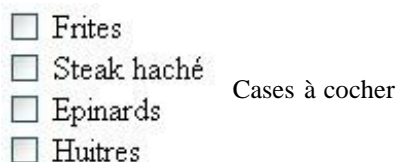
Vous pouvez aussi définir le choix par défaut de la liste. Normalement c'est le premier, mais si vous rajoutez l'attribut `selected="selected"` à une balise `<option>`, alors ce sera le choix par défaut. On pourrait par exemple écrire :

Code : PHP

```
<option value="choix3" selected="selected">Choix 3</option>
```

## Les cases à cocher

La figure suivante représente une série de cases à cocher.



On utilisera le code suivant pour afficher des cases à cocher :

Code : PHP

```
<input type="checkbox" name="case" id="case" /> <label for="case">Ma
case à cocher</label>
```



L'utilisation de la balise `<label>` n'est pas obligatoire mais je la recommande fortement. Elle permet d'associer le libellé à la case à cocher qui a le même `id` que son attribut `for`, ce qui permet de cliquer sur le libellé pour cocher la case. On y gagne donc en ergonomie.

Là encore, on donne un nom à la case à cocher via l'attribut `name` (ici : « case »). Ce nom va générer une variable dans la page cible, par exemple `$_POST['case']`.

- Si la case a été cochée, alors `$_POST['case']` aura pour valeur « on ».
- Si elle n'a pas été cochée, alors `$_POST['case']` n'existera pas. Vous pouvez faire un test avec `isset($_POST['case'])` pour vérifier si la case a été cochée ou non.

Si vous voulez que la case soit cochée par défaut, il faudra lui rajouter l'attribut `checked="checked"`. Par exemple :

**Code : PHP**

```
<input type="checkbox" name="case" checked="checked" />
```

## Les boutons d'option

Les boutons d'option fonctionnent par groupes de deux minimum. Vous trouverez un exemple sur la figure suivante.

Aimez-vous les frites ? ☒ Oui ☐ Non Boutons d'option

Le code correspondant à cet exemple est le suivant :

**Code : PHP**

```
Aimez-vous les frites ?  
<input type="radio" name="frites" value="oui" id="oui"  
checked="checked" /> <label for="oui">Oui</label>  
<input type="radio" name="frites" value="non" id="non" /> <label  
for="non">Non</label>
```

Comme vous pouvez le voir, les deux boutons d'option ont le même nom (« frites »). C'est très important, car ils fonctionnent par groupes : tous les boutons d'option d'un même groupe doivent avoir le même nom. Cela permet au navigateur de savoir lesquels désactiver quand on active un autre bouton du groupe. Il serait bête en effet de pouvoir sélectionner à la fois « Oui » et « Non ».

Pour pré-cocher l'un de ces boutons, faites comme pour les cases à cocher : rajoutez un attribut `checked="checked"`. Ici, comme vous pouvez le voir, « Oui » est sélectionné par défaut.

Dans la page cible, une variable `$_POST['frites']` sera créée. Elle aura la valeur du bouton d'option choisi par le visiteur, issue de l'attribut `value`. Si on aime les frites, alors on aura `$_POST['frites'] = 'oui'`.

Il faut bien penser à renseigner l'attribut `value` du bouton d'option car c'est lui qui va déterminer la valeur de la variable.

## Les champs cachés

Les champs cachés constituent un type de champ à part. En quoi ça consiste ? C'est un code dans votre formulaire qui n'apparaîtra pas aux yeux du visiteur, mais qui va quand même créer une variable avec une valeur. On peut s'en servir pour transmettre des informations fixes.

Je m'explique : supposons que vous ayez besoin de « retenir » que le pseudo du visiteur est « Mateo21 ». Vous allez taper ce code :

Code : PHP

```
<input type="hidden" name="pseudo" value="Mateo21" />
```

À l'écran, sur la page web on ne verra rien. Mais dans la page cible, une variable `$_POST['pseudo']` sera créée, et elle aura la valeur « Mateo21 » !

C'est apparemment inutile, mais vous verrez que vous en aurez parfois besoin.



On croit par erreur que, parce que ces champs sont cachés, le visiteur ne peut pas les voir. C'est faux ! En effet, n'importe quel visiteur peut afficher le code source de la page et voir qu'il y a des champs cachés en lisant le code HTML. Mieux, il peut même modifier la valeur du champ caché s'il a les outils appropriés.

Que faut-il retenir ? Ce n'est pas parce que le champ est caché que vous devez considérer qu'il est inviolable. N'importe quel visiteur (un peu malin) peut le lire, modifier sa valeur et même le supprimer. **Ne faites pas confiance aux données envoyées par le visiteur !** Vous vous souvenez de cette règle dont je vous ai parlé dans le chapitre précédent ? Elle est plus que jamais d'actualité.

## Ne faites jamais confiance aux données reçues : la faille XSS

Vous vous souvenez des mises en garde que j'avais faites dans le chapitre précédent ? Elles ne concernaient pas que les paramètres qui transitent par l'URL : tout cela vaut aussi pour les formulaires !



Mais... autant je vois comment on peut modifier l'URL, autant je ne comprends pas comment peut faire un visiteur pour modifier le formulaire de mon site et trafiquer les données !

On a tendance à croire que les visiteurs ne peuvent pas « bidouiller » le formulaire mais c'est faux. Je vais dans un premier temps vous montrer pourquoi les formulaires ne sont pas plus sûrs que les URL, puis je vous parlerai d'un autre danger important : la faille XSS. Avec ça, nous aurons vraiment fait le tour de ce qu'il faut savoir pour être tranquilles par la suite !

## Pourquoi les formulaires ne sont pas sûrs

Tout ce que nous avons appris dans le chapitre précédent sur les URL reste valable ici. Toutes les informations qui proviennent de l'utilisateur, à savoir les données de `$_GET` et de `$_POST`, doivent être traitées avec la plus grande méfiance.

**Vous ne pouvez pas supposer que vous allez recevoir ce que vous attendiez.**

Cette règle est très simple, mais je vous propose un exemple concret pour bien visualiser le problème. Imaginez que vous demandez à vos visiteurs de rentrer dans un champ leur date de naissance *au format JJ/MM/AAAA*. Combien vont respecter cette mise en forme ? Combien vont se tromper par erreur ? Je vous garantis que parmi tous vos visiteurs, alors que vous attendiez quelque chose comme « 04/10/1987 », vous allez tomber sur une personne qui va écrire : « Je suis né le 4 octobre 1987 ». C'est un exemple un peu extrême mais ça va vous arriver, soyez-en sûrs. Par conséquent, quand vous ferez le traitement de la date en PHP, il faudra bien vérifier qu'elle respecte le format que vous avez indiqué.



Pour vérifier si une chaîne de texte correspond à un certain format, comme « JJ/MM/AAAA », on peut utiliser une validation par *expression régulière*. Les expressions régulières feront l'objet de deux chapitres vers la fin de ce livre car il s'agit d'un sujet assez complexe.

De la même manière, comme dans le chapitre précédent, même si vous demandez un nombre compris entre 1 et 100, il y aura bien quelqu'un pour écrire « 48451254523 ». Soyez donc vigilants et n'ayez jamais confiance en ce qui vient de l'utilisateur, à savoir les données issues des arrays `$_GET` et `$_POST`.

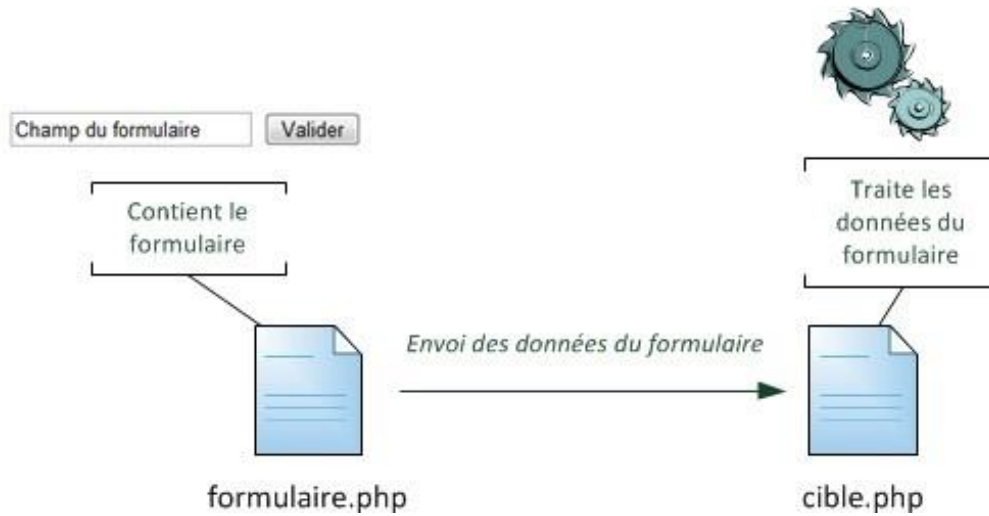
Avec les formulaires, vous ne pouvez pas non plus supposer qu'on va vous envoyer tous les champs que vous attendiez. Un visiteur peut très bien s'amuser à supprimer un champ de texte, et dans ce cas votre page `cible.php` ne recevra jamais le texte qu'elle attendait ! Il faudra impérativement qu'elle vérifie que toutes les données qu'elle attendait sont bien là avant d'effectuer la moindre opération.



Puisque la page du formulaire se trouve sur mon site, comment peut faire un visiteur pour modifier ma page web ? Il peut voir les sources mais pas les modifier !

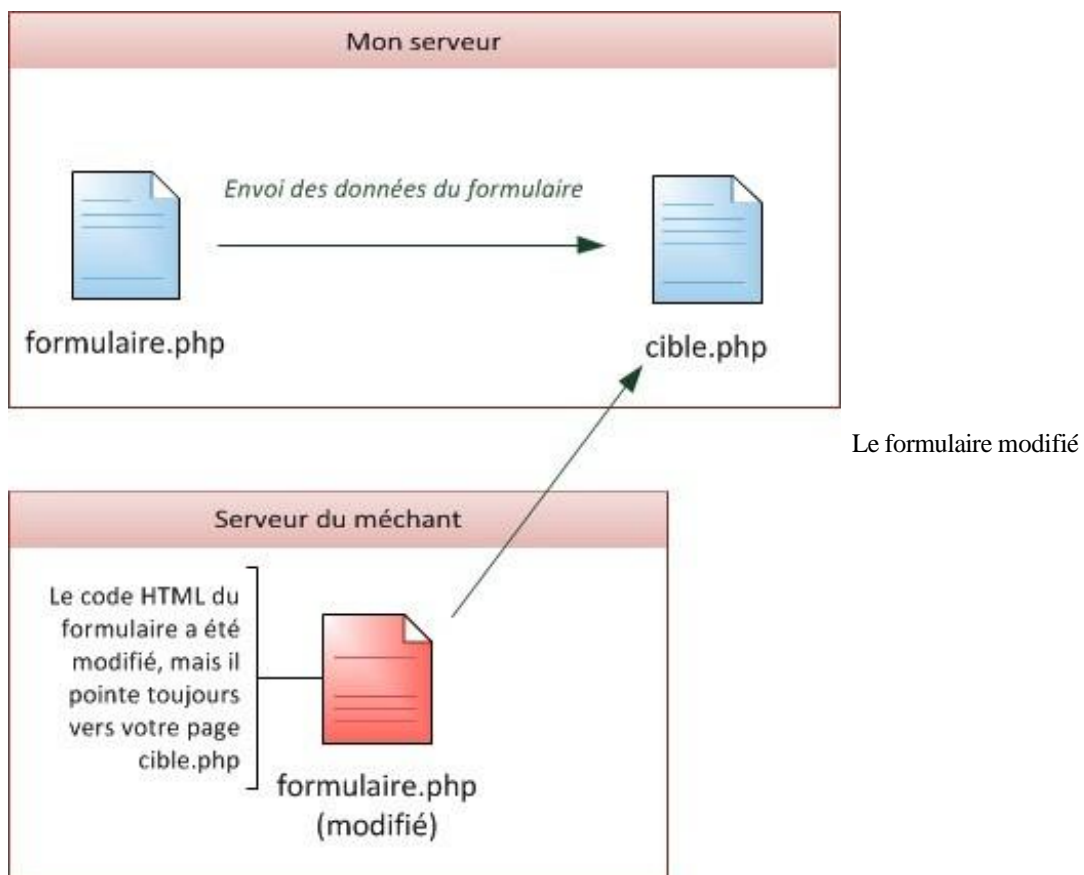
En effet, vos visiteurs ne peuvent pas modifier vos pages web sur le serveur... Mais ils peuvent les reprendre et les modifier ailleurs.

Souvenez-vous du schéma de la figure suivante :



La page `formulaire.php` contient le formulaire et `cible.php` traite les données qu'on lui a envoyées. Autant le code PHP n'est jamais visible par vos visiteurs, autant le code HTML du formulaire, lui, peut être vu par tout le monde.

À partir de là, qu'est-ce qui empêche quelqu'un de créer une copie légèrement modifiée de votre formulaire et de la stocker sur son serveur, à l'image de la figure suivante ?



Sur le schéma de la figure suivante, le « méchant » (nous l'appellerons comme ça, parce que ça lui va bien. ;-)) a pris le code HTML de votre formulaire, l'a modifié et l'a enregistré sur son serveur (ou même sur son ordinateur). L'attribut `action` a été modifié pour indiquer l'adresse absolue (donc complète) de votre page cible :



**Code : PHP**

```
<form method="post" action="http://www.monsite.com/cible.php">
```

Le méchant peut maintenant modifier votre formulaire, ajouter des champs, en supprimer, bref faire ce qu'il veut avec ! Votre page `cible.php` n'y verra que du feu car il est **impossible** de savoir avec certitude de quel formulaire vient le visiteur.

Ces explications sont assez techniques. En fait, on les réserve normalement aux personnes plus expérimentées que les débutants. Cependant, je tenais volontairement à vous montrer comment c'est possible. Même si tout n'est pas totalement clair dans votre tête, vous avez au moins **l'idée** du mode de fonctionnement.

S'il y a une chose à retenir ici, c'est que **les formulaires sont modifiables par tous les visiteurs** contrairement à ce qu'on pourrait penser. Par conséquent, votre page `cible.php` devra être aussi vigilante que nous l'avons été dans le chapitre précédent et ne pas faire confiance aux données de l'utilisateur (les programmeurs ont d'ailleurs une maxime : « **Never trust user input** », ce qui signifie « Ne faites jamais confiance aux données de l'utilisateur »).



Il y a un moyen encore plus simple de modifier le formulaire de votre site sans avoir accès à votre serveur. Internet Explorer 8 et Google Chrome embarquent des « outils pour les développeurs » qui permettent de modifier le code HTML de la page que l'on visite en temps réel. Firefox peut faire de même avec son célèbre plugin Firebug.

## La faille XSS : attention au code HTML que vous recevez !

Il nous reste un dernier point important à voir ensemble et après, j'arrête de vous faire peur, promis !

La faille XSS (pour **cross-site scripting**) est vieille comme le monde (euh, comme le Web) et on la trouve encore sur de nombreux sites web, même professionnels ! C'est une technique qui consiste à injecter du code HTML contenant du JavaScript dans vos pages pour le faire exécuter à vos visiteurs.

Reprenons la page qui affiche le prénom qu'on lui envoie. Elle contient notamment le code suivant :

**Code : PHP**

```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <?php echo  
$_POST['prenom']; ?> !</p>
```

Si le visiteur décide d'écrire du code HTML à la place de son prénom, cela fonctionnera très bien ! Par exemple, imaginons qu'il écrive dans le champ « Prénom » le code : `<strong>Badaboum</strong>`. Le code source HTML qui sera généré par PHP sera le suivant :

**Code : PHP**

```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles  
<strong>Badaboum</strong> !</p>
```



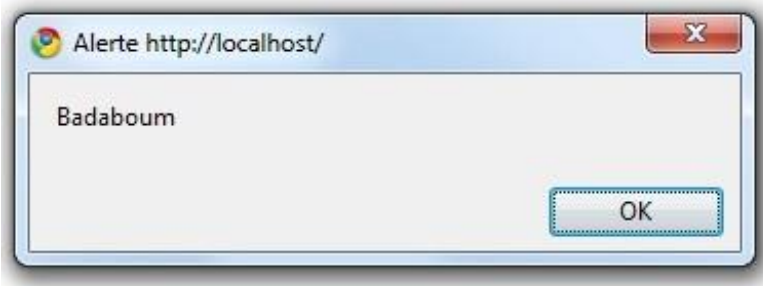
Et alors ? S'il veut mettre son prénom en gras, c'est son problème, non ?

Outre le fait qu'il peut insérer n'importe quel code HTML (et rendre votre page invalide), ce qui n'est pas le plus grave, il peut aussi ouvrir des balises de type `<script>` pour faire exécuter du code JavaScript au visiteur qui visualisera la page !

**Code : PHP**

```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <script  
type="text/javascript">alert('Badaboum')</script> !</p>
```

Tous les visiteurs qui arriveront sur cette page verront une boîte de dialogue JavaScript s'afficher. Plutôt gênant. Voyez la figure suivante.



Exécution de JavaScript par la faille XSS



Les choses deviennent vraiment critiques si le visiteur est assez malin pour récupérer vos cookies de cette façon-là. Les cookies stockent des informations sur votre session et parfois des informations plus confidentielles, comme votre pseudonyme et votre mot de passe sur le site ! Il est possible de forcer le visiteur qui lira le code JavaScript à envoyer tous les cookies qu'il a enregistrés pour votre site web, ce qui peut conduire au vol de son compte sur ce site. Je ne rentrerai pas dans le détail de cette méthode car on s'éloignerait un peu trop du sujet, mais sachez que c'est possible et qu'il faut donc à tout prix éviter qu'un visiteur puisse injecter du code JavaScript dans vos pages.

Résoudre le problème est facile : il faut protéger le code HTML en l'échappant, c'est-à-dire en affichant les balises (ou en les retirant) plutôt que de les faire exécuter par le navigateur, comme sur la figure suivante.

**Mal** : le code HTML a été inséré tel quel dans la page, n'importe quel visiteur peut placer du HTML



Je sais comment tu t'appelles, hé hé. Tu t'appelles **Badaboum** !

Éviter la faille XSS en

Je sais comment tu t'appelles, hé hé. Tu t'appelles <strong>Badaboum</strong> !



**Bien** : le code HTML a été « échappé » et il n'est pas interprété par le navigateur. On voit le HTML mais celui-ci est inoffensif, il ne s'exécute pas.

échappant le HTML

Pour échapper le code HTML, il suffit d'utiliser la fonction `htmlspecialchars` qui va transformer les chevrons des balises HTML `<>` en `&lt;` et `&gt;` respectivement. Cela provoquera l'affichage de la balise plutôt que son exécution.

#### Code : PHP

```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <?php echo  
htmlspecialchars($_POST['prenom']); ?> !</p>
```

Le code HTML qui en résultera sera propre et protégé car les balises HTML insérées par le visiteur auront été échappées :

#### Code : PHP

```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles  
&lt;strong&gt;Badaboum&lt;/strong&gt; !</p>
```



Il faut penser à utiliser cette fonction sur tous les textes envoyés par l'utilisateur qui sont susceptibles d'être affichés sur une page web. Sur un forum par exemple, il faut penser à échapper les messages postés par vos membres, mais aussi leurs pseudos (ils peuvent s'amuser à y mettre du HTML !) ainsi que leurs signatures. Bref, tout ce qui est affiché et qui vient à la base d'un visiteur, vous devez penser à le protéger avec `htmlspecialchars`.



Si vous préférez retirer les balises HTML que le visiteur a tenté d'envoyer plutôt que de les afficher, utilisez la fonction `strip_tags`.

## L'envoi de fichiers

Vous saviez qu'on pouvait aussi envoyer des fichiers grâce aux formulaires ? Vous aurez besoin de lire cette section si vous voulez que vos visiteurs puissent envoyer (on dit aussi **uploader**) des images, des programmes ou tout autre type de fichier sur votre site.



Cette section est un peu plus complexe que le reste du chapitre. Sa lecture n'est d'ailleurs pas obligatoire pour la bonne compréhension de la suite du cours.

Par conséquent, n'hésitez pas à revenir la lire plus tard, lorsque vous en aurez besoin, si vous ne voulez pas vous attaquer de suite à une partie un peu « difficile ».

Là encore, ça se passe en deux temps.

1. Le visiteur arrive sur votre formulaire et le remplit (en indiquant le fichier à envoyer). Une simple page HTML suffit pour créer le formulaire.
2. PHP réceptionne les données du formulaire et, s'il y a des fichiers dedans, il les « enregistre » dans un des dossiers du serveur.



Attention : l'envoi du fichier peut être un peu long si celui-ci est gros. Il faudra dire au visiteur de ne pas s'impatier pendant l'envoi.

On va commencer par créer le formulaire permettant d'envoyer un fichier (une simple page HTML). Nous verrons ensuite comment traiter l'envoi du fichier côté serveur avec PHP.

## Le formulaire d'envoi de fichier

Dès l'instant où votre formulaire propose aux visiteurs d'envoyer un fichier, il faut ajouter l'attribut `enctype="multipart/form-data"` à la balise `<form>`.

#### Code : PHP

```
<form action="cible_envoi.php" method="post"
enctype="multipart/form-data">
  <p>Formulaire d'envoi de fichier</p>
</form>
```

Grâce à `enctype`, le navigateur du visiteur sait qu'il s'apprête à envoyer des fichiers.

Maintenant que c'est fait, nous pouvons ajouter à l'intérieur du formulaire une balise permettant d'envoyer un fichier. C'est une balise très simple de type `<input type="file" />`. Il faut penser comme toujours à donner un nom à ce champ de formulaire (grâce à l'attribut `name`) pour que PHP puisse reconnaître le champ par la suite.

#### Code : PHP

```
<form action="cible_envoi.php" method="post"
enctype="multipart/form-data">
  <p>
    Formulaire d'envoi de fichier :<br />
    <input type="file" name="monfichier" /><br />
    <input type="submit" value="Envoyer le fichier" />
  </p>
</form>
```

Voilà, c'est suffisant.

Vous pouvez ajouter d'autres champs plus classiques au formulaire (champ de texte, cases à cocher). Vous pouvez aussi proposer d'envoyer plusieurs fichiers en même temps.

Là, on va se contenter d'un seul champ (envoi de fichier) pour faire simple.

## Le traitement de l'envoi en PHP

Comme vous avez dû le remarquer, le formulaire pointe vers une page PHP qui s'appelle `cible_envoi.php`. Le visiteur sera donc redirigé sur cette page après l'envoi du formulaire.

C'est maintenant que ça devient important. Il faut que l'on écrive le code de la page `cible_envoi.php` pour traiter l'envoi du fichier.



« Traiter l'envoi du fichier » ? C'est-à-dire ?

Si le fichier a été envoyé sur le serveur c'est bon, non ? Qu'est-ce que PHP aurait besoin de faire ?

En fait, au moment où la page PHP s'exécute, le fichier a été envoyé sur le serveur mais il est stocké dans un **dossier temporaire**. C'est à vous de décider si vous acceptez définitivement le fichier ou non. Vous pouvez par exemple vérifier si le fichier a la bonne extension (si vous demandiez une image et qu'on vous envoie un « .txt », vous devrez refuser le fichier).

Si le fichier est bon, vous l'accepterez grâce à la fonction `move_uploaded_file`, et ce, d'une manière définitive.



Mais comment je sais si « le fichier est bon » ?

Pour chaque fichier envoyé, une variable `$_FILES['nom_du_champ']` est créée. Dans notre cas, la variable s'appellera `$_FILES['monfichier']`.

Cette variable est un tableau qui contient plusieurs informations sur le fichier :

Variable	Signification
<code>\$_FILES['monfichier']['name']</code>	Contient le nom du fichier envoyé par le visiteur.
	Indique le type du fichier envoyé. Si c'est une image gif par exemple, le

<code>\$_FILES['monfichier']['type']</code>	type sera image/gif.
<code>\$_FILES['monfichier']['size']</code>	Indique la taille du fichier envoyé. <b>Attention</b> : cette taille est en octets. Il faut environ 1 000 octets pour faire 1 Ko, et 1 000 000 d'octets pour faire 1 Mo. <b>Attention</b> : la taille de l'envoi est limitée par PHP. Par défaut, impossible d'uploader des fichiers de plus de 8 Mo.
<code>\$_FILES['monfichier']['tmp_name']</code>	Juste après l'envoi, le fichier est placé dans un répertoire temporaire sur le serveur en attendant que votre script PHP décide si oui ou non il accepte de le stocker pour de bon. Cette variable contient l'emplacement temporaire du fichier (c'est PHP qui gère ça).
<code>\$_FILES['monfichier']['error']</code>	Contient un code d'erreur permettant de savoir si l'envoi s'est bien effectué ou s'il y a eu un problème et si oui, lequel. La variable vaut 0 s'il n'y a pas eu d'erreur.



Si vous avez mis un second champ d'envoi de fichier dans votre formulaire, il y aura une seconde variable `$_FILES['nom_de_votre_autre_champ']` découpée de la même manière que le tableau qu'on vient de voir ici.  
`$_FILES['nom_de_votre_autre_champ']['size']` contiendra donc la taille du second fichier, et ainsi de suite.

Je vous propose de faire les vérifications suivantes pour décider si l'on accepte le fichier ou non.

1. Vérifier tout d'abord si le visiteur a bien envoyé un fichier (en testant la variable `$_FILES['monfichier']` avec `isset()` et s'il n'y a pas eu d'erreur d'envoi (grâce à `$_FILES['monfichier']['error']`).
2. Vérifier si la taille du fichier ne dépasse pas 1 Mo par exemple (environ 1 000 000 d'octets) grâce à `$_FILES['monfichier']['size']`.
3. Vérifier si l'extension du fichier est autorisée (il faut interdire à tout prix que les gens puissent envoyer des fichiers PHP, sinon ils pourraient exécuter des scripts sur votre serveur). Dans notre cas, nous autoriserons seulement les images (fichiers .png, .jpg, .jpeg et .gif).  
Nous analyserons pour cela la variable `$_FILES['monfichier']['name']`.

Nous allons donc faire une série de tests dans notre page `cible_envoi.php`.

### *1/ Tester si le fichier a bien été envoyé*

On commence par vérifier qu'un fichier a été envoyé. Pour cela, on va tester si la variable `$_FILES['monfichier']` existe avec `isset()`.

On vérifie dans le même temps s'il n'y a pas d'erreur d'envoi.

#### Code : PHP

```
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas
d'erreur
if (isset($_FILES['monfichier']) AND $_FILES['monfichier']['error']
== 0)
{

}
?>
```

### *2/ Vérifier la taille du fichier*

On veut interdire que le fichier dépasse 1 Mo, soient environ 1 000 000 d'octets (j'arrondis pour simplifier). On doit donc tester `$_FILES['monfichier']['size']` :

**Code : PHP**

```
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas
d'erreur
if (isset($_FILES['monfichier']) AND $_FILES['monfichier']['error']
== 0)
{
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['monfichier']['size'] <= 1000000)
    {

    }
}
?>
```

### 3/ Vérifier l'extension du fichier

On peut récupérer l'extension du fichier dans une variable grâce à ce code :

**Code : PHP**

```
<?php
$infosfichier = pathinfo($_FILES['monfichier']['name']);
$extension_upload = $infosfichier['extension'];
?>
```

La fonction `pathinfo` renvoie un array contenant entre autres l'extension du fichier dans `$infosfichier['extension']`. On stocke ça dans une variable `$extension_upload`.

Une fois l'extension récupérée, on peut la comparer à un tableau d'extensions autorisées (un array) et vérifier si l'extension récupérée fait bien partie des extensions autorisées à l'aide de la fonction `in_array()`.

Ouf ! On obtient ce code au final :

**Code : PHP**

```
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas
d'erreur
if (isset($_FILES['monfichier']) AND $_FILES['monfichier']['error']
== 0)
{
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['monfichier']['size'] <= 1000000)
    {
        // Testons si l'extension est autorisée
        $infosfichier =
pathinfo($_FILES['monfichier']['name']);
        $extension_upload = $infosfichier['extension'];
        $extensions_autorisees = array('jpg', 'jpeg', 'gif',
'png');
        if (in_array($extension_upload,
$extensions_autorisees))
        {

        }
    }
}
```

```

    }
    ?>

```

#### 4/ Valider l'upload du fichier

Si tout est bon, on accepte le fichier en appelant `move_uploaded_file()`. Cette fonction prend deux paramètres :

- le nom temporaire du fichier (on l'a avec `$_FILES['monfichier']['tmp_name']`);
- le chemin qui est le nom sous lequel sera stocké le fichier de façon définitive. On peut utiliser le nom d'origine du fichier `$_FILES['monfichier']['name']` ou générer un nom au hasard.

Je propose de placer le fichier dans un sous-dossier « uploads ».

On gardera le même nom de fichier que celui d'origine. Comme `$_FILES['monfichier']['name']` contient le chemin entier vers le fichier d'origine (C:\dossier\fichier.png par exemple), il nous faudra extraire le nom du fichier. On peut utiliser pour cela la fonction `basename` qui renverra juste « fichier.png ».

#### Code : PHP

```

<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas
d'erreur
if (isset($_FILES['monfichier']) AND $_FILES['monfichier']['error']
== 0)
{
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['monfichier']['size'] <= 1000000)
    {
        // Testons si l'extension est autorisée
        $infosfichier =
pathinfo($_FILES['monfichier']['name']);
        $extension_upload = $infosfichier['extension'];
        $extensions_autorisees = array('jpg', 'jpeg', 'gif',
'png');
        if (in_array($extension_upload,
$extensions_autorisees))
        {
            // On peut valider le fichier et le stocker
définitivement

move_uploaded_file($_FILES['monfichier']['tmp_name'], 'uploads/' .
basename($_FILES['monfichier']['name']));
            echo "L'envoi a bien été effectué !";
        }
    }
}
?>

```



Lorsque vous mettez le script sur Internet à l'aide d'un logiciel FTP, vérifiez que le dossier « uploads » sur le serveur existe et qu'il a les droits d'écriture. Pour ce faire, sous FileZilla par exemple, faites un clic droit sur le dossier et choisissez « Attributs du fichier ».

Cela vous permettra d'éditer les droits du dossier (on parle de *CHMOD*). Mettez les droits à 733, ainsi PHP pourra placer les fichiers uploadés dans ce dossier.

Ce script est un début, mais en pratique il vous faudra sûrement encore l'améliorer. Par exemple, si le nom du fichier contient des espaces ou des accents, ça posera un problème une fois envoyé sur le Web. D'autre part, si quelqu'un envoie un fichier qui a le même nom que celui d'une autre personne, l'ancien sera écrasé !

La solution consiste en général à « choisir » nous-mêmes le nom du fichier stocké sur le serveur plutôt que de se servir du nom

d'origine. Vous pouvez faire un compteur qui s'incrmente : 1.png, 2.png, 3.jpg, etc.



Soyez toujours très vigilants sur la sécurité, vous devez éviter que quelqu'un puisse envoyer des fichiers PHP sur votre serveur.

Pour aller plus loin, je vous recommande de lire le [tutoriel de DHKold sur l'upload de fichiers par formulaire](#) qui traite le sujet plus en détail.

Bonne lecture !

## En résumé

- Les formulaires sont le moyen le plus pratique pour le visiteur de transmettre des informations à votre site. PHP est capable de récupérer les données saisies par vos visiteurs et de les traiter.
- Les données envoyées via un formulaire se retrouvent dans un array `$_POST`.
- De la même manière que pour les URL, il ne faut pas donner sa confiance absolue aux données que vous envoie l'utilisateur. Il pourrait très bien ne pas remplir tous les champs voire trafiquer le code HTML de la page pour supprimer ou ajouter des champs. Traitez les données avec vigilance.
- Que ce soit pour des données issues de l'URL (`$_GET`) ou d'un formulaire (`$_POST`), il faut s'assurer qu'aucun texte qui vous est envoyé ne contient du HTML si celui-ci est destiné à être affiché sur une page. Sinon, vous ouvrez une faille appelée XSS qui peut être néfaste pour la sécurité de votre site.
- Pour éviter la faille XSS, il suffit d'appliquer la fonction `htmlspecialchars` sur tous les textes envoyés par vos visiteurs que vous afficherez.
- Les formulaires permettent d'envoyer des fichiers. On retrouve les informations sur les fichiers envoyés dans un array `$_FILES`. Leur traitement est cependant plus complexe



