

PHP et les formulaires

Sommaire

Introduction.....	3
I. La vérification des données (GET et POST)	4
A. Exemple d'utilisation de la fonction <i>empty</i> effectuée sur une variable <i>\$a non vide</i>	5
B. Exemple d'utilisation de la fonction <i>empty</i> effectuée sur une variable <i>\$b vide</i>	6
C. La négation en PHP	6
II. Autres types de champs	8
A. Explication du message d'erreur.....	10
B. Vérification simultanée des informations transmises	12
C. Le choix de l'opérateur logique	12
D. Conclusion sur la vérification.....	13
III. Organisation du code dans le cas d'un formulaire pointant vers la même page.....	14
IV. À propos de l'utilisation de la méthode GET ou de la méthode POST	18

Crédits des illustrations:
© DR

Les repères de lecture



Retour au chapitre



Définition



Objectif(s)



Espace Élèves



Vidéo /
Audio



Point important /
À retenir



Remarque(s)



Pour aller
plus loin



Normes et lois



Quiz

Introduction

Les formulaires constituent une partie importante du développement PHP. Comme cela a été évoqué dans les pré-requis au début de ce cours, la bonne connaissance des formulaires HTML est requise pour suivre ce chapitre. Vous êtes invités si besoin à réviser le fonctionnement et la mise en oeuvre des formulaires HTML avant de poursuivre ce chapitre.

I. La vérification des données (GET et POST)

Une règle d'or en PHP est de toujours vérifier les données transmises suite à une saisie par l'utilisateur. Ce dernier peut très bien s'être trompé lors de la saisie ou n'avoir pas rempli un des champs.

Les exemples suivants sont réalisés avec GET mais sont tout à fait valables pour POST.

Imaginons que nous souhaitons saluer l'utilisateur par son prénom après que celui ait validé le formulaire. Pour réaliser ce projet, nous allons créer et utiliser les deux fichiers suivants :

- *formulaire.php*
- *resultat.php*

Le fichier *formulaire.php* contient le formulaire HTML :

```
<form action="resultat.php" method="get">
Votre prénom : <input type="text" name="prenom">
<input type="submit">
</form>
```

Fig. 1

Le fichier *resultat.php* est la cible du formulaire qui est définie dans l'attribut *action*. Ce fichier contient le traitement que nous allons effectuer et qui consiste à afficher le prénom saisi par l'utilisateur.

```
<?php
echo 'Bonjour, votre prénom est : ' . $_GET['prenom'] ;
?>
```

Fig. 2

Si l'utilisateur saisi correctement son prénom, le code va permettre d'afficher ce prénom :

```
Bonjour, votre prénom est : Claude
```

Fig. 3

En revanche, si l'utilisateur n'a rien saisi, le code va seulement afficher la chaîne de caractères « Bonjour votre prénom est » que nous avons écrite dans le code.

```
Bonjour, votre prénom est :
```

Fig. 4

Comme nous le voyons, le résultat n'est pas conforme aux attentes car il manque le prénom.

Il ressort de cet exemple que nous devons vérifier si le prénom a bien été saisi par l'utilisateur. Ceci revient à vérifier ce que contient la variable externe.

Afin de bien voir cette variable externe dans le cas où l'utilisateur n'a pas saisi d'informations, nous allons commencer par ajouter un *print_r* à la page resultat.php. Le *print_r* étant une fonction utile seulement pour le développement, il sera supprimé par la suite.

```
<?php
print_r($_GET) ;
echo 'bonjour ' . $_GET['prenom'] ;
?>
```

Fig.5

Le *print_r* nous montre que la variable est vide.

```
Array
(
    [prenom] =>
)
```

Fig.6

Il est donc pertinent de vérifier si la variable est vide avant de vouloir l'afficher dans la phrase de bienvenue débutant par « Bonjour ».

Nous allons poser une structure conditionnelle if/else qui va fonctionner comme suit :

- **si** (*if*) la variable n'est pas vide, nous affichons la phrase de bienvenue et le prénom saisi par l'utilisateur ;
- **sinon** (*else*) nous affichons une phrase qui indique que le prénom n'a pas été saisi.

Pour tester si une variable est vide, nous allons utiliser la fonction PHP *empty*.

A. Exemple d'utilisation de la fonction *empty* effectuée sur une variable *\$a non vide*



Nous testons ici une fonction qui va nous servir dans notre développement. Il arrive souvent que nous devions nous écarter temporairement du projet en cours pour tester le comportement d'une nouvelle fonction c'est-à-dire à bien comprendre son fonctionnement et à maîtriser sa mise en oeuvre. Ensuite nous pouvons reprendre notre projet initial et intégrer cette fonction si elle convient à nos besoins.

Testons donc *empty* :

```
$a = 'il fait beau'; // exemple de variable non vide, qui contient une valeur
if(empty($a)) // si la variable $a est vide
{
    echo 'la variable est vide';
}
else // sinon
{
    echo 'la variable contient une valeur'; // ceci sera donc affiché
}
```

Fig.7

B. Exemple d'utilisation de la fonction *empty* effectuée sur une variable *\$b* vide

Pour déclarer une variable vide il suffit de lui affecter la valeur spéciale null (sans *quote*), qui en PHP représente une variable sans valeur.

Notons que nous aurions pu également affecter deux *quotes* sans espace comme suit *\$b = ''* pour déclarer une variable vide, les deux *quotes* entourent une chaîne de caractères qui n'existe pas dans le cas présent, et donc la variable ne contient aucune valeur.

Cette méthode fonctionne mais est toutefois moins propre que la précédente.

```
$b = null; // exemple de variable vide, qui ne contient aucune valeur
if(empty($b)) // si la variable $b est vide
{
    echo 'la variable est vide'; // ceci sera donc affiché
}
else // sinon
{
    echo 'la variable contient une valeur';
}
```

Fig.8

C. La négation en PHP

Dans la section consacré aux opérateurs de comparaison (chapitre 6 : les opérateurs), nous avons vu que le point d'exclamation sert à marquer la négation. Nous allons voir comment la négation peut nous servir pour concevoir librement notre code.



Au lieu de créer la structure : *Si la variable est vide / Sinon (si elle n'est pas vide)*, nous pouvons utiliser la structure : *Si la variable n'est pas vide / Sinon (si elle est vide)*.

Le code n'est pas fondamentalement différent mais cette dernière structure correspond mieux à la logique de notre vérification.

Bien sûr, nous sommes libres de choisir librement en fonction de nos préférences.

Pour vérifier par exemple qu'une variable **n'est pas vide**, nous créons le code suivant :

```
$a = 'il fait beau';
if(!empty($a)) // si la variable n'est pas vide. Lire NOT EMPTY
{
    echo 'la variable contient une valeur'; // ceci sera donc affiché
}
else
{
    echo 'la variable est vide';
}
```

Fig.9

f. Reprise du projet initial

Nous pouvons désormais reprendre notre projet initial et effectuer la vérification de la variable externe `$_GET['prenom']`.

```
if(!empty($_GET['prenom'])) // si la variable n'est pas vide
{
    echo 'Bonjour votre prénom est ' . $_GET['prenom'];
}
else
{
    echo 'Merci de saisir votre prénom';
}
```

Fig. 10

Ce code fonctionne bien mais nous pouvons remarquer qu'écrire plusieurs fois `$_GET['prenom']` est une tâche relativement lourde et qui peut générer des erreurs à cause du nombre de signes utilisés (dollar, underscore, crochet, quote, etc.). En outre nous devons intégrer cette syntaxe dans des structures conditionnelles qui sont elles-mêmes chargées en signes.

Nous décidons alors de simplifier le code en affectant la variable externe à une variable classique dont la syntaxe plus simple rendra la manipulation plus aisée.

```
$prenom = $_GET['prenom'] ; // assigne la variable externe
```

Fig. 11

Nous pouvons constater qu'il sera nettement plus simple d'utiliser la syntaxe `$prenom` que la syntaxe `$_GET['prenom']` dans notre script.

Le résultat global obtenu est ainsi plus aisé à écrire et à comprendre.

```
$prenom = $_GET['prenom'] ; // création de la variable simple
if(!empty($prenom))
{
    echo 'Bonjour votre prénom est ' . $prenom;
}
else
{
    echo 'Merci de saisir votre prénom';
}
```

Fig. 12

Le fonctionnement de ce script est correct.

Si la variable externe n'est pas vide, l'utilisateur a donc saisi son prénom, et la page affichera :

```
Bonjour votre prénom est Claude
```

Fig. 13

Si la variable externe est vide, l'utilisateur n'a donc pas saisi son prénom, et la page affichera :

```
Merci de saisir votre prénom
```

Fig. 14

II. Autres types de champs

Enrichissons notre formulaire avec d'autres champs destinés à recueillir plus d'informations.

Nous souhaitons connaître le prénom et l'e-mail de l'utilisateur ainsi que sa couleur préférée parmi une liste proposée.

Nous ajoutons donc un champ de type « text » pour la saisie de l'e-mail et une liste de boutons radio associés à des noms de couleurs.

Nous obtenons donc le formulaire suivant :

```
<form action="resultat.php" method="get">
<p><input type="text" name="prenom"> Votre prénom</p>
<p><input type="text" name="email"> Votre email</p>
<p> Votre couleur préférée </p>
<input type="radio" name="color" value="blanc"> Orange <br />
<input type="radio" name="color" value="rouge"> Rouge <br />
<input type="radio" name="color" value="vert"> Vert <br />
<br />
<input type="submit">
</form>
```

Fig. 15

Le formulaire aura cette apparence dans le navigateur :



Fig. 16

Nous allons étudier le cas où tous les champs sont remplis ou cochés par l'utilisateur.

Nous supposons que les valeurs saisies sont : Claude, test@test.com et orange.

En premier lieu nous vérifions dans l'URL que les variables existent bien et ne sont pas vides :

```
http://localhost/essai/resultat.php?prenom=claud&email=test%40test.com&color=orange
```

Fig. 17

Cette première vérification de l'URL est pratique car il suffit de regarder la barre d'adresse du navigateur après avoir validé le formulaire ce qui permet de vérifier d'un coup d'oeil si nous le résultat semble correspondre a priori à nos attentes.

En revanche, consulter l'URL de la barre d'adresse ne permet pas analyse claire et minutieuse des données surtout si plusieurs valeurs sont présentes.



Si cette première lecture de l'URL est pratique, il est cependant recommandé dans tous les projets de systématiquement utiliser un `print_r` ou un `var_dump` pour visualiser les données transmises.

Résultat du `print_r` :

```
Array
(
    [prenom] => claude
    [email] => test@test.com
    [color] => blanc
)
```

Fig. 18

Nous visualisons ici clairement les valeurs transmises pour chacun des champs.

Si nous le souhaitons, il est ensuite simple de récupérer ces valeurs et de les afficher comme vu précédemment :

```
echo $_GET['prenom'] ; //la variable externe contient la valeur du champ prenom
echo $_GET['email'] ; //la variable externe contient la valeur du champ email
echo $_GET['color'] ; / la variable externe contient la valeur du champ color
```

Fig. 19

Étudions maintenant le cas où seul le champ prénom est rempli tandis que le champ e-mail est resté vide et que le choix de la couleur n'a pas été confirmé par l'utilisateur. L'utilisateur a donc saisi uniquement son prénom.

Nous utilisons donc un `print_r` pour visualiser les données transmises :

```
Array
(
    [prenom] => claude
    [email] =>
)
```

Fig. 20



Si un champ de type text n'est pas rempli, il est tout de même transmis dans l'URL mais sa valeur est vide et donc la variable externe correspondante est vide.

Si un champ de type *radio* ou de type *checkbox* n'est pas coché, **il n'est pas transmis** dans l'URL et il n'y aura donc pas de variable externe correspondante.

Nous constatons deux choses :

- la valeur transmise dans l'URL pour le champ email est vide et effectivement la variable externe `$_GET['email']` est vide ;
- **il n'y a pas de valeur transmise dans l'URL pour le champ color** puisque l'utilisateur ne l'a pas coché. Et en effet la variable externe `$_GET['color']` n'existe donc pas.

Si nous gardons l'affichage des trois variables :

```
echo $_GET['prenom'] ;
echo $_GET['email'] ;
echo $_GET['color'] ;
```

Fig. 21

Nous allons alors obtenir l'erreur suivante :


 Notice: Undefined index: color in C:\wamp\www\essai\resultat.php on line 7				
Call Stack				
#	Time	Memory	Function	Location
1	0.0068	669496	{main}()	..\resultat.php:0

Fig. 22

Message d'erreur : la variable externe `$_GET['color']` n'existe pas.

A. Explication du message d'erreur

Le message dit : *Undefined index : color*

Rappelons-nous que la variable externe GET est un tableau et que les clefs d'un tableau sont également appelées des index (ou des indices). L'expression « *index color* » signifie donc la clef *color* du tableau GET.

Pour résumer avec le code ci-dessous

```
echo $ GET['prenom'] ;  
echo $_GET['email'] ;  
echo $_GET['color'] ;
```

Fig. 23

Nous demandons à PHP d'afficher la variable externe `$_GET['prenom']`. Celle-ci existe et contient une valeur, PHP peut donc l'afficher :

- nous demandons ensuite à PHP d'afficher la variable externe `$_GET['email']`. Celle-ci existe mais est vide. PHP n'affiche donc rien ;
- nous demandons ensuite à PHP d'afficher la variable externe `$_GET['color']`. Or celle-ci n'existe pas, et donc PHP ne peut pas l'afficher. Il retourne donc un message d'erreur pour nous signaler que la variable n'est pas définie.

Nous avons donc encore une fois le besoin de vérifier que la saisie de l'utilisateur est conforme à nos attentes. Si l'utilisateur a saisi les informations, nous pourrions les afficher, sinon nous afficherons une phrase qui stipulera que l'utilisateur doit effectuer les saisies demandées.

Nous allons donc **vérifier que les variables ne sont pas vides et qu'elles existent**.

Pour vérifier si une variable est vide, nous savons utiliser la fonction PHP *empty*.

Pour vérifier si une variable existe, nous pouvons utiliser la fonction PHP *isset* qui détermine si une variable est définie.

L'emploi de *isset* est similaire à celui de *empty*.

Par exemple, si nous voulons vérifier qu'une variable `$a` existe :

```
if(isset($a)) // si la variable $a existe  
{  
    echo 'la variable existe';  
}  
else  
{  
    echo 'la variable n\'existe pas';  
}
```

Fig. 24

Dans le cas de notre formulaire, le processus de vérification va donc être fondé sur :

- la vérification que les variables externes issues des champs de type texte **ne sont pas vides** ;

- la vérification que les variables externes issues des champs de type radio **existent**.

```
if(!empty($_GET['prenom'])) // vérification avec empty
{
    echo 'Votre prénom est : ' . $_GET['prenom'] . '<br>'; // variable non vide
}
else
{
    echo 'Vous n\'avez pas saisi votre prénom' . '<br>'; // variable vide
}

if(!empty($_GET['email'])) // vérification avec empty
{
    echo 'Votre email est : ' . $_GET['email'] . '<br>'; // variable non vide
}
else
{
    echo 'Vous n\'avez pas saisi votre adresse email' . '<br>'; // variable vide
}

if(isset($_GET['color'])) // vérification avec isset
{
    // la variable existe
    echo 'Votre couleur préférée est : ' . $_GET['color'] . '<br>';
}
else
{
    // la variable n'existe pas
    echo 'Vous n\'avez pas indiqué votre couleur préférée' . '<br>';
}
```

Fig.25

Les cas sont ainsi tous gérés quelque soient les saisies de l'utilisateur.

Examinons par exemple les deux cas suivants :

Cas n°1 : l'utilisateur a saisi son prénom et choisi une couleur (mais n'a pas indiqué son email)

The screenshot shows a web form with three input fields and a submit button. The first field, labeled 'Votre prénom', contains the text 'Claude'. The second field, labeled 'Votre email', is empty. The third field is a radio button group labeled 'Votre couleur préférée :', with three options: 'Orange' (selected), 'Rouge', and 'Vert'. Below the radio buttons is a button labeled 'Envoyer'.

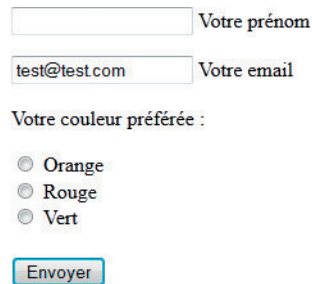
Fig.26

Nous affichons alors :

The screenshot shows the output of the PHP script for Case 1. It displays three lines of text: 'Votre prénom est : Claude', 'Vous n\'avez pas saisi votre adresse email', and 'Votre couleur préférée est : orange'.

Fig.27

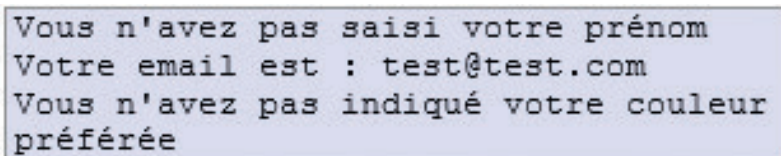
Cas n°2 : l'utilisateur a saisi son email mais n'a pas indiqué son prénom ni sa couleur préférée.



A web form with three input fields and a submit button. The first field, labeled 'Votre prénom', is empty. The second field, labeled 'Votre email', contains the text 'test@test.com'. The third field, labeled 'Votre couleur préférée :', has three radio button options: 'Orange', 'Rouge', and 'Vert', none of which are selected. Below these fields is a blue 'Envoyer' button.

Fig. 28

Nous affichons alors :



A light blue message box with a thin border containing the following text in a monospaced font: 'Vous n'avez pas saisi votre prénom', 'Votre email est : test@test.com', and 'Vous n'avez pas indiqué votre couleur préférée'.

Fig. 29

B. Vérification simultanée des informations transmises

Il arrive fréquemment que nous ne souhaitons afficher aucune information si l'utilisateur n'a pas rempli tous les champs. Dans ce cas nous avons besoin de vérifier **en même temps** que les variables existent et qu'elles ne sont pas vides.

Nous allons donc introduire un opérateur logique dans notre structure conditionnelle *if/else*.

Nous en profiterons pour simplifier la syntaxe de notre code en assignant les valeurs des variables externes à des variables classiques.

C. Le choix de l'opérateur logique

Nous voulons que tous les champs soient remplis ou cochés. Ce besoin peut être traduit comme suit : si la variable externe *prenom* est vide ou si la variable externe *email* est vide ou si la variable externe *color* n'existe pas, alors nous affichons un message rappelant à l'utilisateur qu'il doit saisir tous les champs.

L'opérateur logique que nous utiliserons sera donc **OR** et le code sera donc le suivant :

```
if (
    ( empty($_GET['prenom']) ) // si la variable prénom est vide
    OR
    ( empty($_GET['email']) ) // si la variable email est vide
    OR
    ( !isset($_GET['color']) ) // si la variable color n'existe pas
)
{
    // alors nous avertissons l'utilisateur.
    echo 'Tous les champs du formulaire doivent être remplis';
}
else // sinon tous les champs sont bien remplis
{
    // simplifions la syntaxe en utilisant des variables classiques
    $prenom = $_GET['prenom'];
    $email = $_GET['email'];
    $color = $_GET['color'];

    // affichons les données récupérées
    echo 'Votre prénom est : ' . $prenom . '<br>';
    echo 'Votre email est : ' . $email . '<br>';
    echo 'Votre couleur préférée est : ' . $color . '<br>';
}
```

Fig.30



– Prenez le temps de bien regarder l'imbrication des parenthèses. Les parenthèses du `if` ont été ici coloriées en rouge. Elles englobent l'ensemble des trois conditions.

- Ensuite chaque condition est à son tour entourée de parenthèses afin de bien les distinguer les unes des autres.
- Il est recommandé de sauter des lignes pour bien distinguer les trois conditions, mais vous pouvez préférer les écrire sur une seule ligne. Rappelons-nous que PHP n'est pas sensible aux sauts de ligne ou aux espaces.
- Ensuite, nous avons assigné les variables externes à des variables classiques puis nous avons affiché les informations saisies par l'utilisateur. Le projet est terminé.

D. Conclusion sur la vérification

Il apparaît qu'il est obligatoire de vérifier les données saisies par l'utilisateur. Le premier niveau de vérification que nous avons effectué concerne la saisie ou non des champs du formulaire.

Il existe en revanche d'autres vérifications à effectuer.

Par exemple, l'utilisateur a pu se tromper et indiquer son email à la place de son prénom. Il s'agit dans ce cas d'une erreur d'inattention et ce type d'erreur arrive très fréquemment.

Mais il peut aussi s'agir d'une erreur sciemment produite. N'oublions pas qu'une page d'un site internet est accessible au monde entier et que certains utilisateurs peuvent avoir des comportements peu recommandables. Imaginons par exemple qu'un utilisateur indique une succession de chiffres à la place de son prénom, ou bien qu'il pirate le formulaire

pour transmettre des informations erronées destinées à évaluer notre degré de sécurité.

Il est donc nécessaire de vérifier non seulement que les champs sont bien saisis mais aussi qu'ils contiennent bien les informations attendues.

Par exemple, si nous demandons l'âge de l'utilisateur, nous devons nous assurer que la valeur saisie contient uniquement deux (ou trois) chiffres. Si nous demandons son code postal, nous devons nous assurer que la valeur saisie contient uniquement cinq chiffres et vérifier ensuite que ce code postal existe dans la liste des codes postaux des communes de France. Si nous demandons son adresse email, nous devons nous assurer que la valeur saisie est conforme à la syntaxe permise des adresses emails (par exemple l'email doit contenir un *arobase @*).

La conception d'un formulaire implique que chaque champ fasse l'objet d'une réflexion d'un point de vue vérification.

La vérification des informations transmises par l'utilisateur relève de la gestion de la sécurité d'un site ou d'une application et il s'agit d'un aspect majeur du développement PHP. Nous détaillons ces questions dans le cours PHP de niveau avancé.

III. Organisation du code dans le cas d'un formulaire pointant vers la même page

Un formulaire par le biais de son attribut action indique la page cible vers laquelle nous sommes redirigés après avoir validé ce formulaire. Cette page cible contient le code PHP qui traite les informations transmises. C'est le système que nous avons utilisé jusqu'à présent dans ce chapitre.

Il peut toutefois être pratique que la page contenant le formulaire contienne également le code PHP destiné au traitement. Cela évite notamment la multiplication des fichiers et facilite le développement car toutes les informations HTML et PHP sont ainsi disponibles dans le même fichier.

D'un point de vue HTML, cela ne présente aucune difficulté. Si un formulaire est contenu dans une page *formulaire.php*, il suffit :

- soit de ne pas indiquer l'attribut action

```
<form method="get">
```

Fig.31

- soit d'y indiquer le nom de la page qui contient ce formulaire

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

Fig.32

Pour ces deux syntaxes HTML, la soumission du formulaire nous redirigera vers la page `formulaire.php` qui contient également le code PHP de traitement des informations transmises par ce formulaire.

Voici une première organisation possible d'un formulaire avec un seul champ et un `print_r` destiné à vérifier les informations transmises par le formulaire. Le code PHP est intégré **au-dessus** du formulaire.

```
<?php // début du code PHP
print_r($_GET); // code PHP : ici un print_r
?> // fin du code PHP

<!-- début du code HTML -->
<form action="formulaire.php" method="get">
<p>Votre prénom : <input type="text" name="prenom"></p>
<p><input type="submit"></p>
</form>
```

Fig.33

Cependant lorsque nous consultons cette page, le résultat du `print_r` est affiché même si le formulaire n'est pas validé.

Voici l'affichage du `print_r` au-dessus du formulaire :




Fig.34

Cela signifie que nous devons afficher le `print_r` seulement si le formulaire est validé.

Nous savons que le formulaire est validé si l'URL transmise contient les valeurs saisies.

Il faut dans ce cas stipuler que le résultat du `print_r` sera affiché si la valeur d'un champ est transmise.

Rappelons-nous l'étude que nous avons faite des formulaires : le formulaire contient un champ `prenom` mais aussi un champ de type `submit` qui est le bouton de validation. Ce dernier champ `submit` est également transmis lors de la validation du formulaire (la soumission du formulaire) si nous lui spécifions un attribut `name`.

Cette remarque est importante parce que la meilleure façon au final de déterminer qu'un formulaire a été soumis est de vérifier si le bouton de soumission est transmis avec les autres valeurs.

Ajoutons donc un champ `name` au bouton `submit` du formulaire et validons-le ensuite.

```
<?php
print_r($_GET);
?>
<form action="formulaire.php" method="get">
<p>Votre prénom : <input type="text" name="prenom"></p>
<p><input type="submit" name="validation"></p>
</form>
```

Fig.35

La validation du formulaire va nous donner l'URL suivante :

```
http://localhost/essai/formulaire.php?prenom=claude&validation=valider
```

Fig.36

Nous constatons que cette URL contient bien la variable « validation » (sa valeur « valider » est celle utilisée par défaut par le navigateur car nous ne l'avons pas précisé, celle-ci ne nous étant pas utile).

Nous confirmons ce constat avec le `print_r` qui restitue également cette valeur avec la variable externe `$_GET['validation']`.

```
Array
(
    [prenom] => claude
    [validation] => valider
)
```

Fig.37

Nous allons donc nous servir de la variable `$_GET['validation']` pour déterminer que le formulaire a été validé.

```
<?php
if(isset($_GET['validation'])) // si le formulaire a été validé
{
    print_r($_GET); // alors nous affichons le résultat de print_r
}
?>
<form action="formulaire.php" method="get">
<p>Votre prénom : <input type="text" name="prenom"></p>
<p><input type="submit" name="validation"></p>
</form>
```

Fig.38

Remarquons que nous n'avons pas besoin d'utiliser de `else` dans le cas présent car pour le moment nous cherchons seulement à afficher les informations transmises suite à la soumission du formulaire.

Ainsi les deux cas sont-ils gérés :

Cas n°1 : l'utilisateur consulte la page mais n'a pas encore validé le formulaire.

Votre prénom :

Fig.39

Cas n°2 : l'utilisateur a validé le formulaire. Dans ce cas, la saisie est restituée.

```
Array
(
    [prenom] => Claude
    [validation] => Envoyer
)
```

Votre prénom :

Fig.40

En lieu et place du *print_r* dont nous nous sommes servi uniquement pour préparer notre code, nous pouvons maintenant écrire notre code final en incluant la structure conditionnelle *if/else* de vérification des données saisies par l'utilisateur à l'intérieur de la première structure *if/else* qui vérifie que le formulaire a bien été posté.

```
<?php
if(isset($_GET['validation'])) // si le formulaire est soumis nous vérifions
les champs
{
    if (empty($_GET['prenom'])) // si le champ est vide
    {
        echo 'Tous les champs du formulaire doivent être remplis';
    }
    else // sinon
    {
        // simplifions la syntaxe en utilisant des variables classiques
        $prenom = $_GET['prenom'];

        // affichons le résultat
        echo 'Votre prénom est : ' . $prenom . '<br>';
    }
}
?>
<form action="formulaire.php" method="get">
<p>Votre prénom : <input type="text" name="prenom"></p>
<p><input type="submit" name="validation"></p>
</form>
```

Fig.41

Ainsi les deux cas sont-ils gérés :

**Cas n°1 : l'utilisateur a validé le formulaire sans saisir son prénom.
Dans ce cas nous affichons le message prévu.**

Tous les champs du formulaire doivent être remplis

Votre prénom :

Fig.42

**Cas n°2 : l'utilisateur a validé le formulaire. Dans ce cas nous
affichons correctement sa saisie.**

Votre prénom est : Claude

Votre prénom :

Fig.43

Ce résultat est conforme à nos attentes mais nous pouvons décider que si l'utilisateur a bien saisi et validé son prénom, il n'est alors plus nécessaire d'afficher le formulaire.

Nous allons utiliser la fonction PHP *exit* qui interrompt l'interprétation du script par PHP.

Le code final est donc le suivant :

```
<?php
if(isset($_GET['validation'])) // si le formulaire est soumis nous vérifions
les champs
{
    if (empty($_GET['prenom'])) // si le champ est vide
    {
        echo 'Tous les champs du formulaire doivent être remplis';
    }
    else // sinon
    {
        // simplifions la syntaxe en utilisant des variables classiques
        $prenom = $_GET['prenom'] ;

        // affichons le résultat
        echo 'Votre prénom est : ' . $prenom . '<br>';
        exit; // interrompt le script.
    }
}
/*
Le reste du fichier n'est pas traité par PHP.
le formulaire ci-dessous ne sera donc pas affiché.
*/
}
?>
<form action="formulaire.php" method="get">
<p>Votre prénom : <input type="text" name="prenom"></p>
<p><input type="submit" name="validation"></p>
</form>
```

Fig.44

IV. À propos de l'utilisation de la méthode GET ou de la méthode POST

Le choix de la méthode peut se poser lorsque nous concevons un formulaire. Faut-il choisir la méthode GET ou la méthode POST ?

D'une manière générale, **lorsque le formulaire est en cours de développement**, il est recommandé d'opter pour la méthode GET.

Il sera plus facile de vérifier son fonctionnement notamment grâce à la lecture des données passées dans l'URL.

Ensuite, après avoir bien vérifié que le formulaire fonctionne conformément à nos attentes, nous pouvons à ce moment le modifier et confirmer la méthode qu'il doit posséder qui sera généralement la méthode POST. Il est en effet très souvent demandé que le formulaire fonctionne avec la méthode POST afin justement de ne pas afficher les valeurs dans l'URL, car celles-ci pourraient alors être modifiées par l'utilisateur directement dans la barre d'adresse.

Il peut arriver toutefois que la méthode GET soit retenue, comme par exemple dans le cas du moteur de recherche de Google. Si nous effectuons une recherche dans Google, nous constatons que le formulaire destiné à la recherche fonctionne en effet avec la méthode GET, la preuve en est l'URL transmise par le formulaire qui contient, entre autres données, la valeur saisie dans le champ de recherche.

En résumé, et sauf demande contraire, nous développerons tous nos formulaires avec la méthode POST après les avoir testé avec la méthode GET.



MW0909PV01C

esecad.com