

# PAYE TON PHP

## I-À quoi ça sert ?

Le php n'est rien d'autre qu'une page HTML dynamique. Dynamique ? Avec HTML, c'est bien, tu fais des beaux sites Internet et tout, mais y a un petit souci : Impossible d'interagir avec l'utilisateur.

C'est là qu'intervient le php. En plus des informations qu'on avait déjà avec le HTML, on peut également y inclure des calculs parfois utiles, et surtout pouvoir interagir avec l'utilisateur, et c'est là qu'on va pouvoir utiliser les formulaires de façon utile. Parce qu'HTML, on peut remplir un formulaire, mais après, il va où ? Nul part, puisqu'on ne peut pas utiliser les informations.

## II- Syntaxe

Pour faire du php, on commence par mettre une extension **fichier.php**. Ensuite, comment ça marche ? On écrit sa page web comme d'habitude, on a son doctype, ses balises html et tout ce qui va avec, ce n'est pas ce qui est intéressant en php. Vous le savez sûrement, mais je vais quand même le redire, au cas où : Pour écrire du php, on entre les lignes de code entre des balises bien spécifiques : **< ?php //Mon code php ?>**. D'ailleurs, j'en profite pour vous montrer un commentaire en php.

Ensuite ? Comme en Java, en C++, en C, on entre nos lignes de code, **Séparées par des points-virgules** (On a tendance à les oublier à cause de l'habitude du HTML, mais il faut les mettre. Sinon, ça fait des gentils cadres oranges quand vous chargez les pages (L'acolyte de Mathéo ne le sait que trop bien...), et on en devient vite allergique).

Une variable, on n'a pas besoin de la déclarer en tant que type particulier (genre int, boolean), on la déclare et on lui attribue une valeur. L'ordinateur devine tout seul son type.

```
<p>  
    < ?php  
        $variable = 1; // Ici on initialise $variable à 1, donc c'est un entier.  
    ?>  
</p>
```

« Et comment on l’affiche ? » (Même si le cours l’explique relativement bien) :

```
<p>
    <?php
        $variable = 1; // Ici on initialise $variable à 1, donc c’est un entier.
        echo $variable ;
        echo "La valeur contenue dans la variable est $variable !" ;
        echo 'La valeur contenue dans la variable est '.$variable.' !' ;
    ?>
</p>
```

Quelle différence entre ces trois affichages ? (En particulier les deux derniers)

- Le premier, affichage simple d’une variable, on fait juste echo avec la variable et c’est tout.
- Le second, on veut mettre une phrase avec, histoire de l’introduire et que ça fasse pas trop moche. On met la phrase entre guillemets et on met juste nom de la variable comme si c’était juste du texte (**Sans oublier le \$ quand même...**).
- Le troisième, c’est la même chose que le second, à une différence près : On utilise des apostrophes au lieu des guillemets. Qu’est-ce que ça change ? Si on fait la même chose que pour le deuxième, ça nous affichera « La valeur contenue dans la variable est \$variable ! ». Que faire, alors ? On va utiliser la **concaténation** (« Quel horrible langage », dirait l’Elfe...). Première partie entre apostrophes, ensuite, au lieu d’utiliser le signe +, on va utiliser le point : « . » (Ah bon, vous savez ce que c’est un point ?), on met la variable, on remet un point et on met la fin de la phrase entre apostrophes (Et on oublie pas le point-virgule, Mathéo !).

**ATTENTION :** On ne peut concaténer que des variables de même type. Donc si on concatène un entier avec des chaînes de caractères, le méchant cadre orange viendra nous rendre visite.

## Les tableaux

Voici un exemple de déclaration de tableau (toujours entre les balises php) :

```
$tableau = array(0 => 'Dalek',
                1 => 'Tondeuse',
                2 => 'Escargot',
                3 => 'Jean-François');
```

Je crois qu’il n’y a pas besoin de faire un dessin. Par contre, il y a une chose intéressante. Dans l’exemple, j’ai mis que la case 0 avait pour valeur ‘Dalek’, mais j’aurais très bien pu dire que dans la case ‘Dalek’ il y avait la valeur 0. À partir du moment où c’est le même type d’un côté ou de l’autre pour chaque case du tableau, et qu’on n’a pas deux fois la même valeur dans la première (Vous avez déjà eu deux cases 0 dans un tableau vous ?), on peut faire ce qu’on veut.

## Conditionnelles

Comme en prog, c'est tellement similaire que je vois pas l'utilisé de mettre un exemple. Si ? Bon, d'accord :

```
if ($variable == 1) {  
    echo "La variable vaut 1 bande de salopes !";  
} else {  
    echo "La variable vaut pas 1 bande de salopes ! " ;  
}
```

## Boucles

Etonnement... Pareil qu'en prog. Remarque, c'est normal, php EST un langage de programmation.

```
for ($i = 1; $i < 6; $i++) {  
  
}
```

```
while ($variable == 1) {  
    echo "Je suis dans une boucle infinie." ;  
}
```

## Fonctions

Cette fois on change un peu de la syntaxe habituelle, chaque langage a sa propre façon de créer une fonction (Souvenirs de Scilab pour ceux qui ont fait les oufs en faisant le projet de maths de S1 sur le cryptage RSA...) :

**Déclaration :**

```
function fonctionDeMerde($variable) {  
    echo $variable;  
}
```

**Appel :** fonctionDeMerde(\$variable) ;

## Transmettre des informations

Bon, on entre dans le vif du sujet maintenant. **Comment récupérer des infos d'une page à une autre ?**

Deux méthodes : **GET** ou **POST**

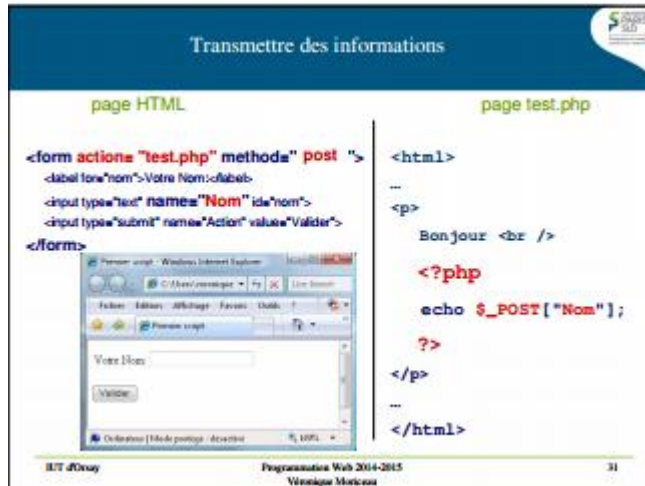
Comment elles fonctionnent ?

- **GET** : À mon sens, elle ne présente pas grand intérêt, dans la mesure où toutes les informations se baladent dans l'URL de la page, donc visibles (Pas tip top pour un mot de passe...), d'autant plus que le nombre de caractères à mettre dans l'URL est limité à 255 (On se doute bien qu'il est préférable qu'un URL ne possède pas trois mille caractères). Même si on arrive rarement à caser 255 caractères juste avec différentes variables, c'est quand même limité.
- **POST** : Plus pratique à utiliser, la confidentialité est respectée, et aucune limite de caractères ! Que du bonheur.

## Dans un formulaire :

Le cours schématise très bien le fonctionnement, du coup je vais le reprendre et m'épargner plusieurs paints.

Avec la méthode **POST** :



Les passages clé sont en rouge. Comment ça marche ?

- **action="test.php"** : C'est le chemin vers la page dans laquelle on va envoyer les informations du formulaire.
- **method="post"** : La méthode qu'on emploie. Soit **get**, soit **post**.
- **name="nom"** : Ici, il faut distinguer les deux parties. C'est dans l'attribut **name** qu'on mettra le nom de la "variable" (Même si c'est pas exactement ça), et c'est ce qu'il y a dans cet attribut que php va utiliser.
- Une fois qu'on a entré son nom ou une connerie dans le cadre correspondant et qu'on a appuyé sur le bouton « Valider », on va s'intéresser à ce qu'il y a dans **test.php**
- **echo \$\_POST["nom"] ;** : Déjà, c'est quoi ce **\$\_POST**, ce truc, là, qui nous emmerde ? C'est une variable qui est déjà prédéfinie dans php. À l'intérieur des guillemets (Ou apostrophes, ça marche aussi), que met-on ? Tout simplement, ce qu'on a mis dans l'attribut **name** de la page avec le formulaire. Si on avait fait **name="M. php est un con"**, déjà ce serait pas très intelligent, et pas très gentil non plus. Ensuite, dans **test.php**, on utilisera **\$\_POST["M. php est un con"] ;**

Voilà, c'est aussi simple que ça.

Avec la méthode **GET** :

Transmettre des informations : GET ou POST ?

- Avec la méthode **GET**, les paramètres seront passés dans l'URL (et donc **visibles** pour l'internaute) :

```
<form action="test.php" method="get">
  <input type="text" name="Nom" id="nom">
  <input type="text" name="Motdepasse" id="motdepasse">
  <input type="submit" name="Action" value="Valider">
</form>
```

**http://monsiteweb/test.php?nom=toto&motdepasse=titi**

IFT d'Orsay      Programmation Web 2014-2015      32  
Vincent Moriceau

- **action= "test.php"** : Je suis vraiment obligé de réexpliquer ?
- **method="get"** : On utilise maintenant la méthode **GET** (Thank you, Captain Obvious !).
- Résultat, comment ça se passe dans l'URL ? On a le nom de la page web qui est créée : **http://monsiteweb/test.php?nom=toto&motdepasse=titi**  
(En passant, j'applaudis l'imagination débordante de nos chers enseignants... Franchement, trouver des noms comme ça, c'est trivial...)
- Et là vous vous dites : « Oh là, oh là, c'est quoi ce bordel ? Pourquoi j'ai ma page php écrite normalement avec ces trucs qui servent à rien derrière ? », ou quelque chose dans le genre (Et si non, ben c'est la même chose, de toute façon je m'en fous, j'ai pas rév... Euh non, je veux dire, je m'en fous, je te fais dire ce que je veux, c'est moi qui fait la fiche, et bordel c'est quoi ça ? Merde à la fin, même plus le droit d'écrire ce qu'on veut ? Et la liberté d'expression t'en fait quoi ? Bref.).
- C'est comme ça que fonctionne **GET** : Ça passe dans l'URL toutes les infos qu'on a mis dans le formulaire.
- Et dans **test.php**, comment on fait pour récupérer ces valeurs ? Non, on ne copie-colle pas ce qu'il y a dans l'URL. Pour la méthode **POST** on avait la variable **\$\_POST["nom"]**, vous vous souvenez ? Et ben là, on remplace juste POST par GET (Ce qui nous donne **\$\_GET["nom"]** pour ceux qui auraient pas suivi), et voilà. « Travail terminé ! », dirait un paysan.

Oui mais voilà, parfois on n'a pas envie d'avoir besoin d'un formulaire, si on veut juste passer une variable d'une page à une autre, comment on fait ?

## Sans formulaire

Mettons que vous avez une variable `$a` qui a pris la valeur 3. Maintenant, vous voulez, pour une raison totalement inconnue, l'afficher sur une autre page, en transitant à l'aide d'un lien (Vous savez, les balises `<a></a>`), et non d'un bouton. Comment peut-on faire ?

En passant par l'URL, comme pour la méthode `GET` :

```
<?php
    $a = 3;
?>
<a href="page2.php?variable=<?php echo $a ?>" method="get">Cliquez ici</a>
```

Explication : On commence par déclarer sa variable `$a` à 3. Même si on referme la balise php, la variable est stockée en mémoire. Ensuite, c'est quoi ce charabia ?

- `<a></a>` : Pas vraiment besoin d'explications
- `href="page2.php?variable=<?php echo $a ?>"` : WATT ?!
  - o `href` : Pas besoin d'explication
  - o `page2.php` : Chemin de la page, pas de nouveauté
  - o `?` : Ça indique qu'il va y avoir des paramètres dans l'URL
  - o `variable=` : Nom avec lequel on va récupérer la valeur de `$a`, j'aurais pu mettre `a=`, ça ne change rien au niveau de la première page.
  - o `<?php echo $a ?>` : Pourquoi ouvrir de nouveau une balise php ? Tout simplement pour mettre le contenu de la variable `$a` dans l'URL (C'est d'ailleurs pour ça qu'on fait `echo`), et non son nom. Si on n'avait pas utilisé la balise php, on se serait retrouvé avec `variable=$a` dans l'URL, et la page suivante, aurait affiché... `$a`.
- `method="get"` : Hé oui, on utilise la méthode get, donc il faut le préciser. Bon, d'accord, si on le met pas ça marche quand même, faites comme vous voulez après tout, c'est votre problème, de toute façon personne ne m'écoute, donc bon...

Est-il possible de faire ça de la même manière qu'avec la méthode **POST** ? La réponse est... Non. Si la question est de savoir si on peut faire ça juste avec un lien, sans bouton, malheureusement, on ne peut pas (Ou je n'ai pas assez cherché). Enfin si, techniquement, on peut, mais ça demande un peu de JavaScript.

La meilleure chose qu'on puisse faire, c'est de faire un formulaire avec des input cachés :

```
<?php $b = "Post"; ?>

<form action="page2.php" method="post">

    <p><input type="hidden" name="variablePost" value="<?php echo $b; ?>" /></p>

    <p><input type="submit" value="Ta mère connard" /></p>

</form>
```

On initialise notre variable qu'on va appeler **\$b**. Ensuite, on crée un formulaire avec la méthode **POST** qui renvoie à la page voulue.

Dedans, comme lorsqu'on fait un cadre pour insérer du texte, on va insérer un objet de type **hidden**. Sa seule fonction sera de stocker une valeur dans l'attribut **value**, où comme précédemment, vous ouvrez une balise php pour afficher le contenu de la variable, et vous refermez. Evidemment, vous pouvez mettre autant de **input** que vous voulez. La seule chose, c'est qu'il faudra obligatoirement un bouton (donc **submit**) pour valider. Sinon, la variable n'est pas retenue dans la page suivante.

Quelques fonctions utiles, probablement rafraichie au cours du temps :

- **isset(\$variable)** : Retourne vrai si la variable passée en paramètre existe. Utile quand on vérifie qu'un **\$\_GET** ou un **\$POST** est bien passé d'une page à l'autre.
- **mt\_rand(entier, entier)** : Sort un nombre aléatoire compris entre le premier et deuxième paramètre.
- **time()** : Si on affiche **echo date("j/m/y, h:i:s", time())**, ça affichera la date et l'heure. Pas en continu, juste l'heure précise (seconde) à laquelle la page a été rafraichie.



## Utiliser des bases de données avec php

La suite au prochain épisode.