PAYE TON PHP

Table des matières

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 chaines 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

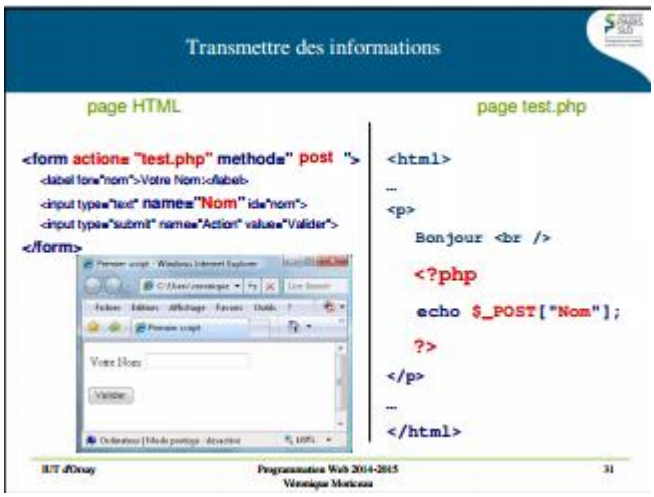
Comment elles fonctionnent ?

- 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 :

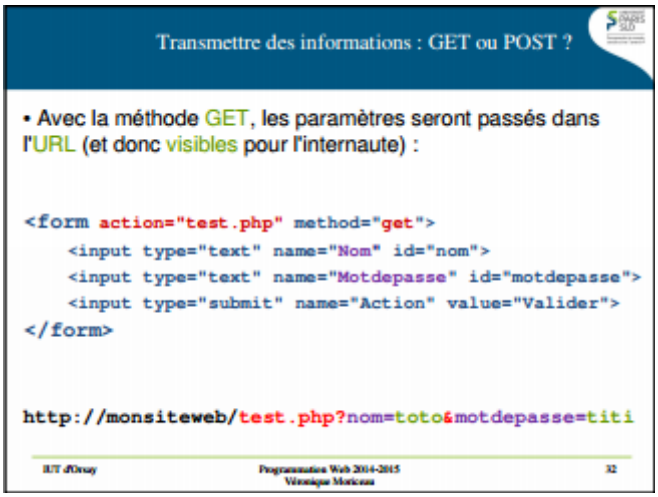


Les passages clé sont en rouge. Comment ça marche ?

- 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 :



- 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 voila. « 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>), 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> : Pas vraiment besoin d'explications
- href="page2.php?variable=<?php echo $a ?>" : WATT ?!
    - href : Pas besoin d'explication
    - page2.php : Chemin de la page, pas de nouveauté
    - ? : Ça indique qu'il va y avoir des paramètres dans l'URL
    - 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.
    - <?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 $POST est bien passé d'une page à l'autre.
- empty($variable) : À la différence de isset, retourne vrai si la variable est vide. Un champ de texte laissé vide existe, donc au lieu d'utiliser la fonction isset, on va utiliser empty.
- 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.
- include(page.php) : inclut un script php. En gros, dans un fichier à part (appelons-le oktamereconnard.php), on écrit que $a = 1 (Le tout dans des balises php). On écrit ensuite dans une autre page include("oktamereconnard.php"); et la magie opère : La variable de la page demandée existe maintenant dans cette page, ainsi que la valeur qu'il y a dedans. Utilité ? Si on veut par exemple affiche un menu de 50 cases sur les 1337 pages php que contiennent notre site, il suffit juste d'écrire ça une fois dans un fichier à part, et ensuite l'inclure dans toutes les pages.
- header('Location: chemin/de/la/page.php') : Fonction de redirection. Imaginez que vous avez un formulaire d'inscription, et que vous vérifiez la validité. Dans verification.php vous allez commencer par appeler cette fonction **avant tout code html**. Vous faites ensuite votre vérification, et vous appelez une nouvelle fois cette fonction (à l'identique). Attention à ne pas mettre d'espace avant les deux points, sinon ça ne fonctionne pas.

Utiliser des bases de données avec php

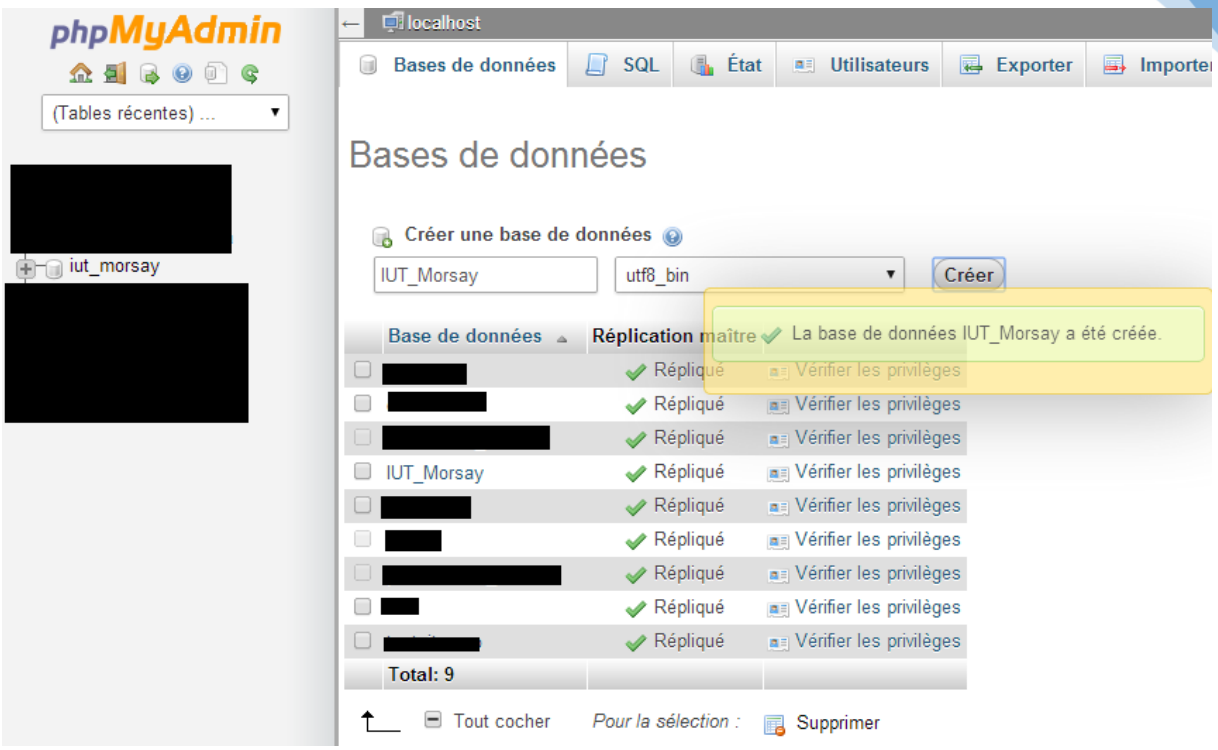
phpMyAdmin

Avant toute chose, on va commencer par l'ouvrir. Non, ce n'est pas un logiciel de plus à télécharger. Quand vous ouvrez vos pages php, vous tapez « localhost » dans la toolbar et vous naviguez entre vos… Bon, autant montrer une image, ce sera plus rapide.

Voilà, vous avez trouvé phpMyAdmin. Maintenant, quand vous cliquez dessus, normalement vous arrivez à une page d'identification (Je dis normalement, parce que j'utilise WampServer et non XAMPP, et l'affichage est un peu différent). Vous tapez « root » pour l'utilisateur, et Rien Du Tout pour le mot de passe, et vous validez. J'espère que ça ne sera pas trop dur à retenir. Dans le doute, faites-vous un pense-bête (Il portera bien son nom, pour une fois ;-)).

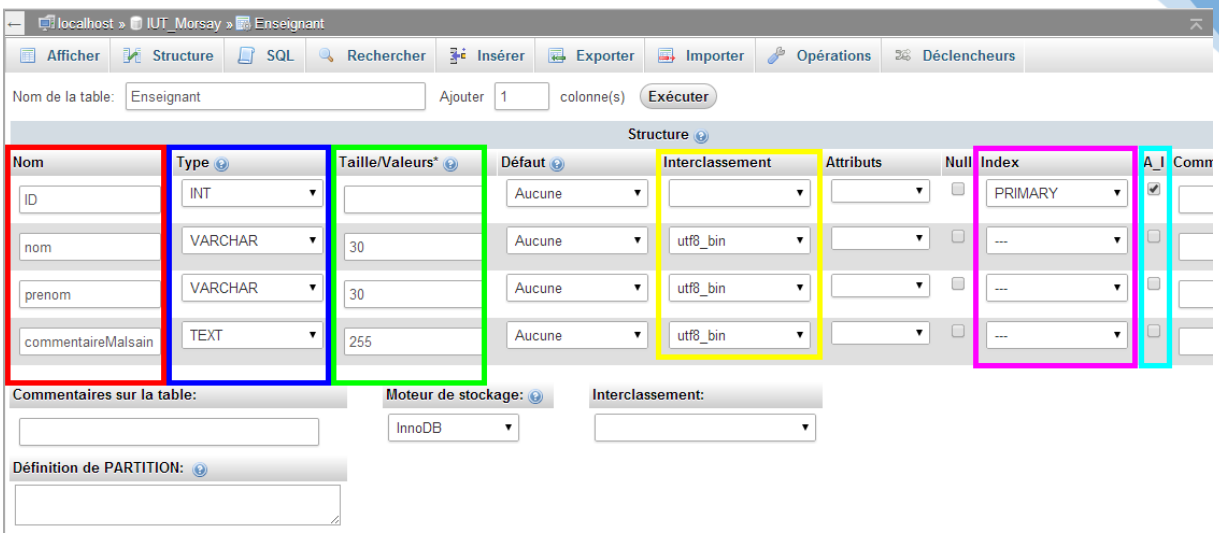
Pour la suite, les images que je vais montrer risque de heurter la sensibilité des plus jeunes. Non, en fait, elles seront juste un peu différentes de ce que vous aurez vous quand vous suivrez avec attention cette fiche, pour la même raison que ce que j'ai dit dans le paragraphe précédent. Ce sera un peu différent, mais globalement, ça change rien.

Attaquons. On va créer notre première base de données (C'est chiant à écrire cette connerie, à partir de maintenant j'abrégerai par bdd). Pourquoi pas l'appeler « Camping » ? Avec des tables « BaseLoisirs », « Compocamping » et… Non ? Bon, d'accord. Appelons-la « IUT_Morsay ».



Je vous avais dit que l'affichage serait un peu différent. Voilà, j'ai créé ma bdd IUT_Morsay, mais c'est quoi ce « utf8_bin » ? Quand on code en HTML, on utilise l'encodage UTF-8. Ici, on appelle ça l'interclassement. Je vous conseille de mettre cet interclassement à chaque fois. Après, vous pouvez ne pas le faire. Y en a qui ont essayé, ils ont eu des problèmes. Si tout n'est pas en adéquation (Vazy comment t'utilises des mots compliqués !), vous pourrez dire adieu à vos accents et autres caractères spéciaux.

Maintenant qu'on a créé notre bdd, on va y ajouter des tables, sinon elle sert pas à grand-chose. On clique donc sur notre nouvelle bdd, et on va nommer notre première table, ainsi que son nombre de colonnes. Faites bien attention au nombre de colonnes ! Si vous en mettez trop, c'est pas trop grave, mais s'il vous en manque, vous ne pourrez pas en rajouter par la suite. On va commencer par l'offensive : La table Enseignant. Dedans, on aura son ID, son nom, son prénom, et un commentaire malsain à son sujet. Quatre colonnes, donc.



Les choses importantes sont encadrées (Comment ça on s'en serait douté ?)

- Nom : Le nom de la colonne, je vais pas vous faire un dessin.
- Type : Est-ce que vous voulez que votre colonne soit un entier, une chaine de caractères etc. Les plus courants sont INT (entier), VARCHAR (chaine de caractères), TEXT (Vous êtes si nul que ça en anglais ?), et DATE (Ça aussi, c'est assez explicite).
- Taille/Valeurs : Utile uniquement pour ce qui contient des caractères, c'est la taille maximale que prendra votre attribut. Faites donc attention de ne pas mettre trop petit.
- Interclassement : Encore et toujours le même problème, mettez le même pour chacun.
- Index : Quel type de clé est-ce ? Ici, notre ID est une clé primaire.
- A.I : C'est quoi ce truc ? Ce truc ? C'est plus noir, plus intense, plus… *crock*. A.I Signifie Auto Increment. Ça veut dire que vous n'aurez pas à préciser sa valeur à chaque nouvelle valeur dans votre table. Votre ID sera auto-incrémenté.

  Pour le reste, j'avoue ne pas encore avoir trouvé d'utilité.

  Je ne vous fais pas une démonstration de « Comment ajouter une ligne dans votre table, je suis certain que vous trouverez tout seul, comme un grand (Ou une grande, si vous êtes de la gente féminine).
  On peut aussi passer directement par une requête SQL pour modifier ses tables à volonté. Pour cela, on clique sur l'onglet SQL et on entre sa requête. Personnellement, je ne l'utilise qu'en cas d'extrême nécessité (Genre si j'ai 50 lignes à rentrer, je copie-colle un INSERT avec ce que je veux, c'est quand même plus rapide que d'ajouter une ligne à chaque fois).
  Je vous laisse aussi le plaisir d'apprendre par vous-même à supprimer une table.

  Maintenant que la partie chiante est terminée, on va attaquer la partie intéressante : Comment fait-on pour accéder à notre bdd avec php ?

Accéder aux bdd avec php

Je vais commencer par entrer les lignes de code, après j'explique. Je vais finalement faire la méthode du cours, tout simplement parce que celle que j'ai montré précédemment risque d'être supprimée d'ici quelques années. Mais rassurez-vous, quasiment rien ne change.

```
<?php
        $co = mysqli_connect("localhost","root","","IUT_Morsay") or die ("erreur de connexion");
        mysqli_query($co, "SET NAMES UTF8");
?>
```

C'est quand même plus beau avec des couleurs !
Explication :

- `$co = mysqli_connect('localhost','root','',"IUT_Morsay")` : Vous aurez probablement deviné aux paramètres. Cette fonction permet de se connecter au serveur (localhost, puis les identifiants root et mot de passe vide, et enfin le nom de la bdd). Pourquoi ce `$co` ? C'est la variable qui sera utilisée pour savoir de quelle connexion à quelle bdd on parle (Si on se connecte sur deux serveurs en même temps, c'est pratique d'avoir deux variables distinctes).
- `or die ("erreur de connexion");` : Si la connexion à votre serveur échoue, votre page affichera erreur de connexion à la place du cadre orange allergisant.
- `mysqli_query("SET NAMES UTF8");` : Pour faire apparaitre vos accents correctement, il faut que l'encodage soit le même pour tout, c'est-à-dire la bdd, la connexion vers la bdd, le script php et la page html. Ici, c'est la connexion à la bdd qu'on met en UTF8. Voilà l'utilité de cette fonction.

Pour se déconnecter, il suffit d'écrire ceci : `<?php mysqli_close($co); ?>` à la fin de votre script, ce qui signifie qu'on ferme la connexion à ce serveur.

Comment écrire une requête SQL en php ? On a deux méthodes, quasi similaires : Soit on écrit la requête dans une chaîne de caractères, et on appelle la fonction miracle, soit on appelle directement la fonction miracle avec la chaîne de caractères directement dedans. Personnellement, je préfère procéder en deux étapes.

```
$requete = "SELECT * FROM IUT_Morsay ORDER BY nom";
$res = mysqli_query($co, $requete) or die("Requête invalide");
```

- Je commence par stocker dans ma variable `$requete` la requête SQL.
- Puis, dans une autre variable, `$res`, je stocke le résultat de cette requête (Sans oublier de préciser la connexion `$co`). Pourquoi ? Parce qu'après, on va récupérer chaque ligne du résultat grâce à cette variable.
- Pour exécuter la requête, il faut donc appeler la fonction `mysqli_query($co, $requete)`, avec toujours la possibilité que la requête soit invalide, et éviter un cadre orange.

Remarque : La requête SQL peut être n'importe quelle requête, aussi bien d'insertion que de suppression, de modification ou de sélection.

Afficher ou stocker le résultat d'une requête

Maintenant qu'on a effectué notre requête SQL et qu'on a sélectionné toutes les lignes de notre table IUT_Morsay, il va falloir qu'on puisse l'afficher. On peut toujours afficher la variable, mais je ne pense pas que ça fonctionne, et quand bien même ça fonctionnerait, ça ferait un truc moche. Je n'ai même pas osé essayer.
Il y a deux possibilités à ce que vous voulez faire : Afficher ou stocker ces valeurs ? Il y a très peu de différences, à vrai dire.

Voilà ce que ça donne, si vous voulez simplement afficher votre requête :

```
while ($row = mysqli_fetch_assoc($res)) {
        echo "L'enseignant n°".$row['ID']." s'appelle ".$row['prenom']." ".$row['nom']." !<br />
        En voici une appreciation tout à fait objective : ".$row['commentaireMalsain']."<br />
        <br />";
}
```

Pour simplifier, ça fonctionne un peu comme un foreach : Pour chaque valeur du résultat de la requête, qu'on stocke dans la variable $row, on exhibe ses attributs.
La partie à retenir : `while ($row = mysqli_fetch_assoc($res))`

Si vous voulez stocker les valeurs de la requête dans des variables :
Il faudra d'abord initialiser des tableaux avant d'entrer dans la boucle. `$mot = array();` etc.

```
while ($row = mysqli_fetch_assoc($res)) {
        $id[] = $row['ID'];
        $nom[]  = $row['nom'];
        $prenom[]  = $row['prenom'];
        $descr[] = $row['commentaireMalsain'];
}
```

La partie importante : Je vais prendre exemple avec $id mais c'est valable pour les quatre. Lorsqu'on y attribue la valeur de $row, il ne faut pas oublier les crochets [] pour préciser qu'on rajoute une ligne et qu'on y met cette valeur.

Vérifier les formulaires

Expression régulière

Faire des formulaires, c'est bien, mais éviter que l'utilisateur fasse de la merde avec, c'est mieux. Pour éviter que quelqu'un mette « oktamereconnard..lol_-^@ù%98/*.enfoiré » en guise d'adresse mail, on peut limiter ce qu'ils écrivent en utilisant des **expressions régulières**. C'est quoi encore cette connerie ? Vous vous rappelez les automates, en maths graphes ? Avec les alphabets, toutes ces choses qu'on a eu la joie d'oublier pendant les vacances d'été ? On va réutiliser le principe. Bon, un exemple : Si je vous dis (a|b)*, ça veut dire quoi ? Qu'on va écrire a ou b autant de fois qu'on veut. Ça y est, ça vous revient ? Bien.

Maintenant, on va faire ça en langage php. Déjà, toute expression régulière… Oh flemme, un exemple et j'explique après. Ceci est un exemple d'expression régulière :
^[a-zA-Z]*[0-9]{2,6}\.n+$

Oh, c'est beau toutes ces couleurs ! C'est magique, et ça sera utile pour la compréhension, enfin la lecture.

- ^ : **Indique que le mot commence par ce qu'on veut vérifier**
- [a-zA-Z] : On veut une lettre de l'alphabet, minuscule ou majuscule, donc de a à z (a-z) ou de A à Z (A-Z). Les caractères spéciaux ne sont pas compris dedans. **Aucun espace**
- * : Indique qu'on veut la sélection précédente autant de fois qu'on veut (0..* en CPO). **Il y a d'autres signes : + = 1..* et ? = 0..1**
- [0-9] : Vous l'aurez compris, on veut seulement un chiffre compris entre 0 et 9.
- {2,6} : Ici, on veut l'expression précédente entre 2 et 6 fois.
- \. : Antislash pour les caractères particuliers, donc ici on veut que ce soit suivi d'un point.
- n : On veut un n, tout simplement. Pourquoi j'ai mis ça dans l'exemple ? Pour montrer qu'on peut juste demander un (ou plusieurs) caractère(s) spéci(al/aux) dans une expression régulière. Et puis je fais ce que je veux ? C'est quoi ces manières ?!
- + : Pareil qu'au-dessus.
- $: **Indique que le mot finit par ce qu'on veut vérifier**

Donc dans l'exemple si dessus, on veut une suite de lettres, suivie de chiffres (entre 2 et 6), ensuite il doit y avoir un point et une suite de « n » (Au moins 1). **Oktamereconnard01.nnn**, par exemple, remplit les conditions précédentes, et est donc valide.

Quelques exemples pratiques :

- Un code postal : ^[0-9]{5}$
- Une date (jj-mm-aaaa) : ^[0-9]{2}-[0-9]{2}-[0-9]{4}$
- Un mot commençant par une majuscule suivit de lettres minuscules : ^[A-Z][a-z]*$

Utiliser avec php

C'est bien beau de savoir faire une expression régulière, mais maintenant il va falloir l'utiliser pour vérifier ses formulaires (au risque de me répéter).

Donc on a notre formulaire avec un seul champ (j'ai pas dit que ce serait un formulaire utile) de texte avec mail comme name. Et bien sûr, on utilise la méthode post, ça me parait évident.
Dans un premier temps, il vaut mieux vérifier que le champ n'est pas vide. Pour cela, on va utiliser la fonction empty($_POST["mail"]). Si elle est vide, je vous laisse le choix du traitement. Sinon, il faut vérifier qu'elle soit valide.

Imaginons qu'on veuille une adresse mail qui commence par une lettre minuscule, suivie par des lettres ou des chiffres, suivie du signe @, suivie de lettres, suivie d'un point, suivie d'une extension à 2 ou 3 caractères minuscules.

Je vous laisse chercher l'expression régulière pendant que je l'appelle $expression_reguliere (Non, je me suis pas fait chier). Pour comparer la valeur du mail avec ce qu'on veut, on va utiliser une nouvelle fonction :

```
preg_match($expression_reguliere, $_POST["mail"]) ;
```

Cette fonction renvoie vrai si le champ comparé est bien valide par rapport à notre expression régulière.

Toujours pas trouvé ? Bon, voilà la correction :

```
$expression_reguliere = "#^[a-z][a-zA-Z0-9]*@[a-zA-Z]*\.[a-z]{2,3}$#";
```

Vous devez avoir remarqué quelque chose… Rien ? C'est que vous n'êtes vraiment pas doué… J'ai englobé toute l'expression régulière de #. Une expression régulière basique n'en a pas besoin, mais en php, les expressions régulières doivent être délimitées par le symbole #.

Petit récap' sur les expressions régulières :

- ^ et $ indiquent respectivement le début et la fin d'une expression régulière

- [aeiouy] : Indique la présence **obligatoire** d'**une** des lettres présentes entre les crochets

- [a-zA-Z0-9] : Pareil qu'au-dessus, mais au lieu de lettres précises, ce sont des listes de lettres ou chiffres

- [^a-z] : La présence du ^ indique qu'on ne veut pas de chiffre (ou quelconque caractère qui suit)

- ?, +, * : Le caractère (ou choix entre crochets précédant l'un de ces symboles sera répété respectivement 0 ou 1 fois, 1 ou n fois ou 0, 1 ou n fois

- {2}, {2,6}, {2,} : Présent 2 fois, entre 2 et 6 fois, ou 2 fois et plus

- \ : Symbole utilisé pour rendre vérifiable les caractères spéciaux : `# ! ^ $ ( ) [ ] { } ? + * . \`

- Classes abrégées (Un peu comme des raccourcis) :
  - \d : Tous les chiffres. Équivalent à [0-9]
  - \D : Tout sauf les chiffres. Équivalent à [^0-9]
  - \w : Caractère alphanumérique ou underscore. Équivalent à [a-zA-Z0-9_]
  - \W : Tout sauf un mot. Équivalent à [^a-zA-Z0-9_]
  - \t : Tabulation
  - \n : Nouvelle ligne
  - \r : Retour chariot
  - \s : Espace blanc
  - \S : Tout sauf un espace blanc

- Le signe « . » : N'importe quel caractère sauf \n

- / ! \ J'ai dit tout à l'heure que le symbole « # » servait à délimiter les expressions régulières, mais j'ai menti. Devant le second dièse, il est possible d'ajouter une option. Deux exemples :
  - #expressionReguliere#i : Rend insensible à la casse (Maj ou min, on s'en fout)
  - #expressionReguliere#s : Rajoute la classe abrégée \n au symbole « . »

Il est bien entendu possible de cumuler les options : #expressionReguliere#is est valide.

Formulaires PHP et sécurité

Vous avez vos inscrits, leurs mots de passe, et à l'intérieur leur numéro de carte bancaire, ainsi qu'une photo de leur chien. Il vaut mieux que certaines informations restent confidentielles, et c'est pourquoi il est question de sécurité. Et qui dit sécurité dit faille (Oh le méchant !). Il y a plusieurs types de failles que les pirates exploitent, et il vaut mieux s'en protéger. On va en voir deux (même s'il en existe beaucoup plus, on n'a vu que ces deux-là en cours, donc on ne parlera que de celles-là) : les failles XSS et les injections SQL. Ce sont des vilains mots, mais on va les expliquer après, bien sûr.

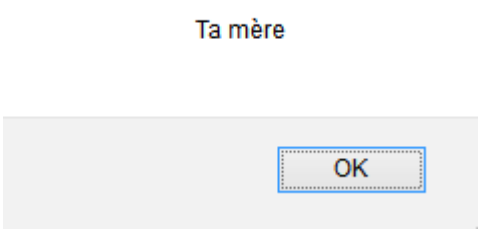
Failles XSS

Je vais reprendre ce que dit le TD, c'est très bien expliqué.

La faille XSS est utilisée pour injecter des codes malveillants en langage de script (comme Javascript) dans des sites Web. Les scripts sont alors exécutés côté client (par le navigateur de l'internaute).

Comment l'exploite-t-on ?

Vous avez votre formulaire avec le login, le mot de passe et le bouton de connexion. Ce bouton renvoie à une autre page qui récupère le login et qui dit bonjour à celui-ci. Jusque-là, rien d'inquiétant. Oui mais voilà, on peut mettre n'importe quoi dans le login. Imaginons que vous écrivez ceci : `<script>alert("Ta mère")</script>`, dans le champ du login. Certains penseront que c'est un login de merde. D'autres y verront un petit problème. Moi non plus, je n'aime pas trop les balises `<script>`, encore moins ce qu'il y a dedans. C'est en fait un code en Javascript qui ouvre une boîte de dialogue. Ainsi, quand vous cliquez sur « Valider », vous aurez votre nouvelle page, mais une boîte de dialogue dans laquelle sera marqué Ta mère s'ouvrira aussitôt.



Si vous avez essayé et que ça ne fonctionne pas, ce n'est pas un problème de recopiage ou quoi, c'est juste que votre navigateur a empêché la boîte de dialogue de s'ouvrir. Quand je l'ai essayé, Chrome ne faisait rien, mais Firefox générait la boîte de dialogue. Un petit affichage comme ça, ce n'est pas gênant, mais si vous vous y connaissez un peu, vous sauriez probablement quoi mettre comme alerte pour faire crasher le site.

Comment empêcher ça ?

Il existe plusieurs fonctions qui permettent ça :

- $string = htmlspecialchars($_POST['login']) : Convertit les caractères spéciaux en entités HTML. C'est flou ? Un exemple : le symbole < sera remplacé par < dans le code source (empêchant ainsi le script de s'activer), et l'affichage sera <script>alert("Ta mère")</script>.

- $string = htmlentities($_POST['login']) : En plus de la fonction précédente, elle convertit tous les caractères relatifs au codage HTML ou Javascript. Donc plus ou moins la même chose que précédemment.

- $string = strip_tags($_POST['login']) : Supprime les balises. Plus radical, c'est pas possible. Du coup, votre <script>alert("Ta mère")</script> deviendra alert("Ta mère"), ce qui empêche également le script d'être généré.

Voilà, vous savez maintenant vous défendre contre les pirates informatiques débutants qui ne savent exploiter que les failles XSS. Il faudra probablement les utiliser pour le projet, gardez-le donc en mémoire.

On va donc passer au deuxième type d'attaque : les injections SQL.

Injections SQL

Un pirate saisit du code SQL dans un formulaire qui, si le formulaire n'est pas vérifié, va s'injecter dans une requête SQL définie par le site Web. La requête d'origine est alors modifiée et l'injection peut permettre d'afficher des données cachées, d'écraser des valeurs, de modifier la base de données, etc.

Vous avez donc votre formulaire avec login, mot de passe et bouton « valider », ainsi que votre toute nouvelle base de données dans laquelle vous avez créé un utilisateur : sacré_ITOPeuh, avec le mot de passe oktamereconnard.rar. Votre formulaire est envoyé à une page qui vérifie si l'utilisateur existe. Enfin, on affiche « Bonjour » suivi du login.

Vous avez donc quelque part la requête suivante :

```
SELECT login, motdepasse FROM utilisateurs WHERE login = \"".$_POST['login']."\" AND motdepasse = \"". $_POST['mdp']."\"
```

Si vous n'avez pas compris, vous vérifiez que le login et le mot de passe que vous avez entré sont bel et bien dans la table. On ne s'intéressera ici qu'à la partie mot de passe :

```
AND motdepasse = \"". $_POST['mdp']."\"
```

Maintenant, dans votre formulaire, dans le champ du mot de passe, vous écrivez ceci :

```
" OR 1=1#
```

Vous allez me dire « C'est quoi ce truc de merde ? », et quand vous vous connectez, pouf, ça marche. Et là, vous ne comprenez pas. On va remplacer le $_POST['mdp'] par ce qu'il contient :

```
AND motdepasse = "" OR 1=1#"
```

(J'ai retiré les "/" pour plus de lisibilité)

Décortiquons un peu :

- " : Ferme les premiers guillemets, la requête vérifie donc un mot de passe nul : `motdepasse = ""`

- OR 1=1 : La requête vérifie que 1=1. Sauf si on est vraiment con, c'est toujours vrai. Vu que c'est précédé d'un OR, la condition sera vraie pour toutes les lignes de la table.

- # : Cette petite subtilité permet de mettre en commentaire le guillemet de derrière qui est en trop.

Et voilà, pour peu qu'on connaisse le login de plusieurs personnes, on va pouvoir accéder à leurs données personnelles grâce à cette technique.
Et là vient la question fatidique : Comment empêcher ça ?

Il existe également une fonction spécifique à ce type d'attaque :
mysqli_real_escape_string($co, $_POST['mdp']) : Protège les caractères spéciaux d'une chaîne pour l'utiliser dans une requête SQL, en prenant en compte le jeu de caractères courant de la connexion.

Bon, mieux vaut une petite démonstration :

```
echo "Mot de passe original : ".$_POST['mdp']."<br />\n
Mot de passe protégé : ".mysqli_real_escape_string($co, $_POST['mdp'])."<br />";
```

En gros, j'affiche d'abord le mot de passe original, et ensuite j'affiche ce que la fonction modifie.
Et voilà le résultat :

Mot de passe original : "OR 1=1#

Mot de passe protégé : \"OR 1=1#

La fonction a changé le mot de passe pour qu'il soit protégé des injections (Je ne rentre pas dans les détails).
Pour vous protéger des injections SQL, je vous propose une solution :
Une fois que la connexion à la base de données est faite, avant de faire la moindre requête SQL, vous commencez par vérifier que le mot de passe original et le mot de passe protégé sont identiques. Oui, oui, je vous montre.

```
if (mysqli_real_escape_string($co, $_POST['mdp']) != $_POST['mdp'] {
        echo 'Tu vas trop loin. Moi et ma famille, ça ne nous fait pas rire.' ;
} else {
        /* Vous pouvez vérifier que l'utilisateur existe etc. */
}
```

Vous voilà protégés contre les failles XSS et les injections SQL ! Youpi !

Le php Objet

Vous l'attendiez tous, nous allons maintenant attaquer le php objet ! Vous savez au moins ce qu'est un objet non ? public class MachinTruc {}, MachinTruc easy_medias = new MachinTruc() ; La mémoire vous revient ? Bien.
On ne va pas passer vingt minutes sur une introduction de « Pourquoi le php Objet ? », mais juste préciser une ou deux choses. Il est vivement conseillé de créer ses classes dans des fichiers externes et de les inclure après. C'est même la chose la plus logique à faire en fait, je ne devrais même pas avoir à vous le dire.
Passons directement à la racine du « problème » : La syntaxe.

Syntaxe

On déclare une classe de la façon suivante (bien évidemment à déclarer dans des balises php) :

```
class GateauChocolat {
        /* Tout plein de choses */
}
```

C'est tout con hein ? Passons aux attributs (Pas ceux-là !) :

```
private $pasteque1;
public $pasteque2;
protected $pasteque3;
```

Vous voyez, c'est pas compliqué. Bien entendu, l'attribut protected sous-entend qu'on peut avoir des classes mères et des classes filles (avec extends), et on peut également faire des classes abstraites, des interfaces (Et donc implements), et tout le tagada-tsoin-tsoin possible du Java.

Bon, maintenant qu'on a notre classe GateauChocolat et nos trois pastèques (qu'on mettra uniquement en private, c'était juste pour l'exemple), il faut bien l'instancier quelque part. Et qui dit instance dit constructeur ! Un exemple de constructeur :

```
public function __construct($pasteque1, $pasteque2, $pasteque3) {
        $this->pasteque1 = $pasteque1;
        $this->pasteque2 = $pasteque2;
        $this->pasteque3 = $pasteque3;
}
```

On va maintenant décortiquer comme on l'a toujours fait :

- public function __construct : Comme on peut le deviner, c'est une fonction qui agit par défaut comme un constructeur. / ! \ **Particularité** : Il faut mettre ***DEUX*** « _ » avant construct, sinon il y a une erreur de syntaxe.

- ($pasteque1, $pasteque2, $pasteque3) : Tout simplement les arguments de la fonction, il n'y a rien de particulier ici.

- $this->pasteque1 = $pasteque1; : Là, ça devient intéressant :
    - **$this** : Comme le this de Java
    - **->** : Équivalent au « . » en Java, on peut dire que $this pointe vers son attribut, même si je pense que c'est pas correct de dire ça comme ça.
    - **pasteque1** : Nom de l'attribut. / ! \ **Attention** : Le signe « $ » se met devant « this », et c'est tout. $this->$attribut n'est pas correct.
    - = $pasteque1; : Là, pas de nouveautés, on attribue à l'attribut (Hoho, jeu de mots, malynx le lynx !) la valeur de l'argument, et on n'oublie pas le point-virgule.

- Et voilà, on a un constructeur tout beau tout propre. Faites quand même attention à l'ordre des arguments, comme il n'y a pas de type explicite.

- / ! \ **Attention** : Dans **toutes** les fonctions et procédures, quand on veut utiliser un attribut, il faut faire $this->attribut. Si on fait juste $attribut, il cherche une variable locale qui n'existe pas, et c'est pas bon.

« Oui mais si je veux plusieurs constructeurs ? »
Tu peux pas. Et puis tu m'emmerdes avec tes questions. Mmm ? Bon, d'accord.

Imaginons qu'on veuille, pour une raison lambda, plusieurs constructeurs d'une même classe. Komenkékonfai ? Et là, vous allez tous me répondre à l'unisson :
« Ben on fait plusieurs fonctions __construct() avec un nombre d'arguments différent à chaque fois ! »
Sauf que ça ne marche pas comme ça. La fonction __construct(), on la déclare une seule fois, et c'est tout. Du coup, c'est une autre méthode qu'il faut employer.

```
public function __construct() {
        /* Avec la fonction func_num_args(), on récupère le nombre d'arguments */
        switch(func_num_args()) {
                case 1 : /* Un paramètre */
                $this->pasteque1 = func_get_arg(0);
                /* On attribute à notre première pastèque le premier argument */;break;

                case 2 : /* Deux paramètre */;break;
                case 3 : /* Trois paramètre */;break;
                default : break;
        }
}
```

Je pense qu'il est inutile d'expliquer point par point, les commentaires s'en chargent très bien tous seuls.

Ainsi, on a deux façons de faire notre constructeur : Soit vous savez qu'il n'y aura jamais qu'un seul constructeur, et que vous êtes sûr de vous, et vous utilisez un seul et unique constructeur, soit vous ne savez pas de combien de constructeurs vous allez avoir besoin, ou au contraire vous savez qu'il en faudra plus d'un, et vous utilisez un **constructeur sans arguments** et avec un switch pour gérer tous les cas que vous voulez. C'est vous qui décidez.

Bon, maintenant qu'on a notre constructeur, il serait peut-être temps que notre classe serve à quelque chose ! Qui dit **classe** dit **fonctions** et **procédures**.

```
public function getPasteque1() {
        return $this->pasteque1;
}
```

Voilà, on a un **getter** de notre première pastèque. C'est donc une **fonction**.
Et un setter, vous me direz ? Quasiment la même chose, en fait.

```
public function setPasteque1($pasteque1) {
        $this->pasteque1 = $pasteque1;
}
```

Hé oui. Une procédure n'est rien d'autre qu'une fonction sans return. Au moins, pour s'en souvenir, c'est plus facile.
Nous avons donc notre classe, il va maintenant falloir l'appeler à partir d'une page, sinon elle ne sert pas à grand-chose. Voici un exemple qui résume bien les opérations possibles :

```
include("GateauChocolat.php");/* Inclusion de notre classe */
$magik = new GateauChocolat("La réponse de la vie ?",42); /* Création de notre objet */
$magik->setPasteque1(23);/* On modifie notre pastèque avec le setter qu'on a créé juste avant */
echo "Tout se rapporte à ce nombre : ".$magik->getPasteque1();/* Affichage avec le getter */
```

Cookies et sessions

Cookies

C'est quoi un cookie ? À quoi ça sert ? Ça se mange ? Pleins de questions auxquelles je vais répondre.

- Qu'est-ce qu'un cookie, et à quoi ça sert ? C'est une variable qui garde en mémoire des informations pendant un temps donné. C'est flou ? Ne vous inquiétez pas, ce sera plus clair avec un exemple.

- Ça se mange ? Oui.

Bon, imaginons que vous ayez une page formulaire.php un formulaire qui contient un champ nom, un champ prénom et un bouton qui envoie ses informations par la méthode post à la page verification.php. Normalement, verification.php vérifie les informations envoyées (Thank you, Captain Obvious !), mais là on ne va pas le faire, c'est pas le but. Ce qu'on veut, c'est que quand on revient (automatiquement) sur le formulaire, les champs du nom et du prénom ne soient pas effacés. On va donc, dans verification.php, créer des cookies et leur attribuer les valeurs des noms et prénoms. Ne paniquez pas si vous ne comprenez pas, l'exemple fera la différence en termes d'explications.

verification.php

```
<?php   header('Location: formulaire.php');
        setcookie('nom', $_POST['nom'], time()+10);
        setcookie('prenom', $_POST['prenom'], time()+10);
        header('Location: formulaire.php');
?>
```

Bon, comment ça marche ?

- header('Location: formulaire.php'); : Toujours en début de code, avant toute autre chose. **Faites bien attention à NE PAS METTRE D'ESPACE AVANT LES DEUX POINTS**. Oui, je parle en connaissance de cause.

- setcookie('nom', $_POST['nom'], time()+10); : Là, ça devient intéressant :
  - setcookie : Méthode pour créer un cookie.
  - 'nom' : Premier paramètre, ce sera par ce nom qu'on appellera le cookie.
  - $_POST['nom'] : Second paramètre, c'est la valeur qu'on attribue au cookie. C'est comme cela qu'on crée une mémoire.
  - time()+10 : Troisième paramètre, il s'agit de l'heure à laquelle expirera le cookie. On peut aussi dire que 10 est la durée du cookie (en secondes).

- header('Location: formulaire.php'); : On renvoie à la page principale.

Maintenant qu'on a compris comment créer des cookies, il va falloir savoir comment les utiliser et les appeler. Pour cela, rien ne vaut un exemple :

formulaire.php

```
<!DOCTYPE html>
<html>
    <head>
    </head>

    <body>
        <form method="post" action="verification.php">
            <p><label for="nom">Nom : </label><input type="text" name="nom"
id="nom" value="<?php if (isset($_COOKIE['nom'])) echo $_COOKIE['nom']; ?>" /></p>
            <p><label for="prenom">Prénom : </label><input type="text"
name="prenom" id="prenom" value="<?php if (isset($_COOKIE['prenom'])) echo
$_COOKIE['prenom']; ?>" /></p>
            <p><input type="submit" value="Valider" /></p>
        </form>
    </body>
</html>
```

Bon, beaucoup de code pour pas grand-chose, à vrai dire. La partie qui nous intéresse se trouve ici :

```
value="<?php if (isset($_COOKIE['nom'])) echo $_COOKIE['nom']; ?>"
```

Explication :

- value= : Valeur que contiendra par défaut le champ

- if (isset($_COOKIE['nom'])) : C'est comme ça qu'on appelle un cookie. De la même façon qu'on appelle un $_POST. On vérifie donc qu'il existe (Et donc qu'il a bien été créé).

- echo $_COOKIE['nom']; : On place la valeur que contient le cookie dans le champ du formulaire.

Si vous avez bien tout suivi, ce cookie survivra 10 secondes, puis disparaitra. Ainsi, les informations que vous voulez garder resteront actives pendant le temps que vous avez décidé dans les variables $_COOKIE['nomDuCookie'] **sur toutes les pages par lesquelles vous transitez.** Si par exemple vous avez un lien qui vous envoie vers une page rang d'homme, et que dans cette page vous appelez les variables $_COOKIE['nom'] et $_COOKIE['prenom'], elles seront utilisables, toujours pendant le temps que vous leur avez prédéfinies.

/ ! \ Il faut créer ses cookies avant tout code html, sinon ça ne fonctionne pas

Sessions

C'est quoi une session ? Sur un site où on a besoin de se connecter (et entrer ses identifiants), dès lors qu'on change de page, toutes les informations sont perdues, et on doit de nouveau entrer ses identifiants. Une première solution : les cookies. On stocke les informations précises dans des cookies, et elles sont donc gardées en mémoire, et l'utilisateur n'a pas besoin de retaper ses identifiants à chaque nouvelle page. Problème : les cookies restent un temps limité. Même si on dit qu'elles doivent durer 10 minutes, on ne sait pas si l'utilisateur va rester trente secondes ou une demi-heure. C'est donc trop peu précis. Solution : On créer des variables de sessions. Elles fonctionnent plus ou moins de la même manière qu'un cookie, la différence réside dans le fait que sa durée est celle que passe l'utilisateur sur le site.

Comment ça marche ?

Avant toute chose, sur chacune des pages de votre site, vous devez appeler la fonction session_start() **avant tout code html (même avant le DOCTYPE).** C'est cette fonction qui va créer une session quand un visiteur se connectera. **Toutes les pages doivent commencer par cette fonction**. Autrement, vous ne pourrez plus accéder à ces variables.
La seconde chose est qu'après avoir créé une session, il faut la supprimer à la fin de la visite d'un utilisateur. On utilise pour cela la fonction session_destroy(). Elle sera appelée lorsque vous demanderez à l'utilisateur de se déconnecter, mais **elle est également appelée automatiquement** lorsque l'utilisateur ne charge plus aucune page du site pendant plusieurs minutes (timeout).

Voici les différentes étapes d'utilisation des sessions.

1- Un visiteur arrive sur le site et entre ses identifiants. Une session lui est alors créée. Cette session possède un identifiant unique (très gros et écrit en hexadécimal) qui lui est propre. Cet identifiant est généré automatiquement par php.

2- Une fois la session créée, on peut créer des variables de session (autant qu'on veut) relatives à l'utilisateur. On crée une variable de session de la façon suivante :
```
$_SESSION['nomDeLaVariable'] = valeur ;
```
On peut stocker n'importe quel type de variable à l'intérieur d'une variable de session.

3- Une fois que l'utilisateur se déconnecte, on détruit la session avec la fonction session_destroy() et toutes les informations créées à partir de celle-ci seront également détruites.

Bon, la partie explication est désormais terminée, on va passer à la partie application. Vous avez une page formulaire_session.php où il y a un formulaire nom/prénom/bouton qui redirige vers la page session.php qui traite ces informations. À partir de cette session, vous avez un lien qui vous place sur la page page_interne.php, qui dit souhaite la bienvenue au prénom envoyé. *Si vous avez bien suivi les cours, vous remarquerez que c'est exactement le TD8 du cours*. Bon, on va rajouter un petit truc : Dans la page page_interne.php, on va rajouter un lien qui va sur lui-même et qui déconnecte la session. Comme ça on verra bien que toutes les informations relatives à la session ont été supprimées.

Je ne vais pas réécrire le formulaire, il est tout ce qu'il y a de basique. C'est du HTML à 100%.

session.php

```php
<?php   session_start();
        if (!empty($_POST['nom']) && !empty($_POST['prenom'])) {
                echo "Bienvenue ".$_POST['prenom']."<br />";
                $_SESSION['prenom'] = $_POST['prenom'];
?>
<a href="page_interne.php">Cliquez ici</a>
<?php
        } else {
                echo "Nom ou prénom non remplis";
        }
?>
```

Explication :

- <?php session_start(); : On crée dès le début la session.
- Simple vérification du prénom et du nom. Si la vérification est bonne, on crée la variable de session « prenom » avec la ligne $_SESSION['prenom'] = $_POST['prenom'];
- On crée ensuite le lien vers la page interne
- Si la vérification n'est pas bonne, on l'indique.

Voilà pour session.php. Globalement, ça n'a pas rien d'insurmontable.

Maintenant, on va regarder du côté de page_interne.php

```php
<?php   session_start();
        if (isset($_SESSION['prenom'])) {
                $prenom = $_SESSION['prenom'];
                echo "Bienvenue $prenom";
        } else {
                echo "prenom inconnu";
        }
?>
<p><a href="page_interne.php" onclick="<?php session_destroy(); ?>">Se déconnecter</a></p>
```

Alors, comment ça marche ?

- Toujours pareil, on appelle la fonction session_start().
- Ensuite, on vérifie que la variable de session du prénom existe (C'est le cas puisqu'on l'a créée dans session.php), et si c'est le cas on dit « Bienvenue » avec le prénom.
- Si elle n'existe pas (Donc si la session est « fermée » (détruite est le terme plus exact)), on dit qu'il y a un problème.

- Maintenant, intéressons-nous à cette dernière ligne :
    - : Pas de problème de ce côté-là.
    - <a href="page_interne.php" onclick="<?php session_destroy(); ?>"> :
        - Dans un premier lieu, la redirection vers elle-même. Pas de problème.
        - C'est quoi ce onclick= ? C'est ce que vous décidez que la page fera au clic de ce lien. Comme vous pouvez le constater, lorsqu'on clique sur le lien, on appelle la fonction session_destroy(), qui détruira la session. C'est, en html, un argument très pratique pour des sites dynamiques.
    - Après, vous connaissez.

Si vous avez bien compris (ou si vous avez essayé (le code, hein, pas essayé de comprendre)), vous verrez que quand on clique sur Se déconnecter, la session est détruite, et l'affichage change (Vu que la variable de session est perdue).

Voilà, vous savez maintenant à peu près tout ce qu'il y faut savoir sur les cookies et les sessions en php.

MVC

Je ne m'étalerai pas trop sur ce sujet, simplement que c'est une arborescence de fichiers qui est la plus utile (et logique) à utiliser.

MVC : Modèle/Vue/Contrôleur

Ce sera dans ces trois fichiers que seront répartis tous vos fichiers php.

- Vue : Ici, il y aura tous les fichiers relatifs à l'affichage. Toutes les pages que l'utilisateur est susceptible de voir.

- Modèle : Ici, il y aura toutes les pages qui seront appelées depuis le modèle. Il y aura par exemple la connexion à la base de données, les différentes classes, les différentes fonctions etc.

- Contrôleur : Ici, il y aura toutes les pages qui traitent l'information et qui contrôlent le fonctionnement principal du site. C'est par le biais de celles-ci que transiteront les informations entre le modèle et la vue. Toutes les pages de la vue appelleront d'autres pages de la vue ou des pages du contrôleur, mais **jamais celles du modèle**.

Que dire de plus ? Bon, on va reprendre le TD9 du cours pour illustrer :

- Controleur
  - Connexion.php
  - Deconnexion.php
  - Inscription.php
- Modele
  - Bd.php
  - Membre.php
- Vue
  - Espace_membre.php
  - Formulaire_connexion.php
  - Formulaire_inscription.php

C'est assez clair : La vue affiche ce que l'utilisateur doit voir, le modèle contient les classes (C'est dans la classe bd.php qu'on aura la connexion à la base de données), et le contrôleur contient toutes les pages qui vérifient, valident et redirigent.

Bonne chance à tous !