



Secteur Tertiaire Informatique
Filière étude - développement

Développement Web

Développer des pages Web Dynamiques en PHP

Accueil

Mise en pratique

Période en
entreprise

Evaluation



Code barre

SOMMAIRE

1. Objet des exercices	4
2. Premier exercice : hello world !	4
3. Deuxième exercice : variables d'environnement	7
Date du jour	8
utilisation de l'instruction getenv()	8
parcours du tableau associatif \$_SERVER	9
4. Troisième exercice : les variables de session	9
5. Quatrième exercice : exploitation des formulaires HTML	15
Formulaire HTML de saisie	15
Page de réponse	16
script formul.php :	16
Variante pour gérer les cases à cocher :	18
6. Bilan des exercices	20
Fondamentaux	20
Syntaxe PHP	20
Variables de session	20
Exploitation des formulaires côté serveur	20

1. OBJET DES EXERCICES

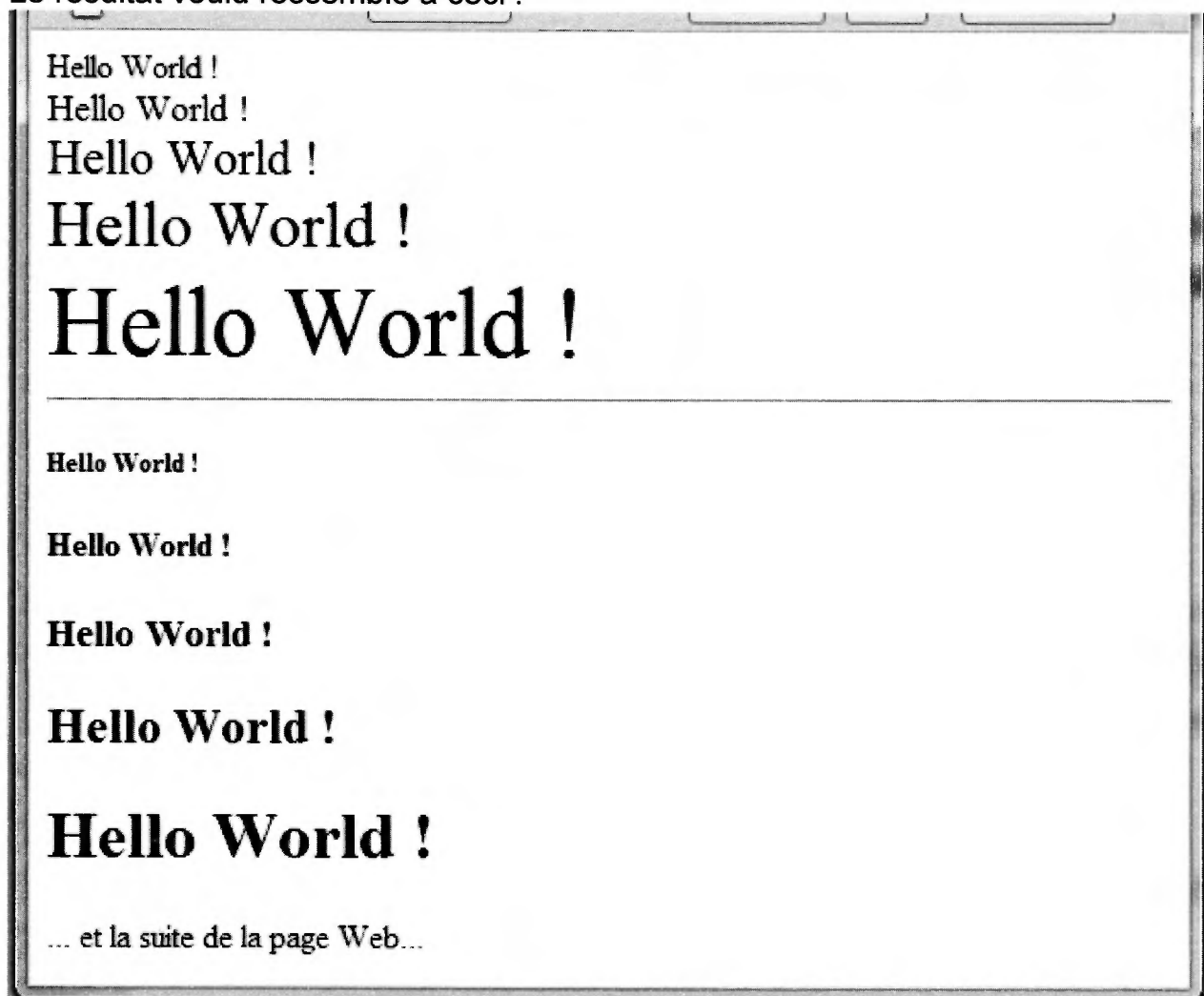
Il s'agit ici d'une série d'exercices guidés visant à comprendre les techniques de développement de pages Web dynamiques à l'aide du langage PHP.

- **EXO1** Hello Word acquisition des premiers éléments du langage PHP.
- **EXO2** Connaître les variables d'environnement et les tableaux associatifs
- **EXO3** Travailler avec les variables de Session et comprendre leur rôle.
- **EXO4** Exploitation des formulaires.

2. PREMIER EXERCICE : HELLO WORLD !

Ce premier exercice est une variante du programme d'affichage du message « Hello World ! » utilisé fréquemment pour présenter les techniques de programmation de chaque langage.

Le résultat voulu ressemble à ceci :



Le code HTML correspondant à la première partie pourrait être :

```
1
2 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
3 <html>
4
5 <head>
6 <meta http-equiv="Content-Type" content="text/html;
  charset=utf_8">
7 <title>Une boucle for next pour dire Bonjour au monde</title>
8 </head>
9
10 <body>
11 <font size="3">Hello World !</font><br />
12 <font size="4">Hello World !</font><br />
13 <font size="5">Hello World !</font><br />
14 <font size="6">Hello World !</font><br />
15 <font size="7">Hello World !</font><br />
16 <hr />
```

Bien sûr, un développeur pourrait écrire ce code HTML tel quel grâce à quelques copier/coller mais nous allons ici faire générer une partie de ce code HTML par PHP, puisque c'est la raison d'être du langage PHP !

On remarque que, dans la partie répétitive, seule la valeur de l'attribut *size* varie de ligne en ligne ; il serait donc judicieux de faire varier automatiquement une variable depuis la valeur 3 jusqu'à la valeur 7 et d'injecter cette valeur dans un « *flot HTML* » construit « *à la volée* ».

Le véritable langage de programmation PHP dispose de tout l'attirail nécessaire : déclaration et utilisation de **variables**, instructions de **tests**, instructions **répétitives** comme la boucle *for* adaptée à notre cas.

Rappel des principes PHP :

- dans le code PHP, toute portion de code HTML/CSS/JavaScript écrit en dehors des balises PHP spécifiques sera envoyée sans modification au navigateur demandeur ;
- toute portion de code PHP écrite dans une balise spécifique <?php ...?> sera interprétée et son résultat inclus dans le flot HTML en lieu et place de la balise PHP ;
- les scripts PHP doivent être enregistrés avec une extension particulière (.php par défaut) pour être interprétés par le serveur Web Apache/PHP avant leur envoi au navigateur.

Nous pouvons donc obtenir le code HTML ci-dessus grâce au script PHP suivant :

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf_8">
<title>Une boucle for next pour dire Bonjour au monde</title>
</head>
<body>
<?php for ($i = 3; $i <= 7; $i++)
{
    echo '<font size="' . $i . ">Hello World !</font><br />\n'; // syntaxe classique
}
print("<hr />\n"); // ou echo "<hr />\n";
?>
```

code HTML constant

boucle PHP de type for

Génération ligne de code HTML avec concaténation et caractères spéciaux

ne pas oublier la suite du code HTML de manière à produire une page HTML complète et valide

Reproduire et tester en jouant cette page, **servie en HTTP** par votre serveur Web Apache/PHP.

Variantes :

Pour mieux comprendre la syntaxe PHP, saisir ces variantes et tester le résultat :

```
<body>
<?php for ($i = 3; $i <= 7; $i++)
{
    echo '<font size="' . $i . ">Hello World !</font><br />\n';
    echo "<font size=\"$i\">Hello World !</font><br />\n";
    echo '<font size=$i>Hello World !</font><br />\n'; // ou
}
print("<hr />\n"); // ou echo "<hr />\n";
```

concaténation implicite de la valeur de la variable \$i

erreurs : la variable \$i et le caractère spécial ne sont pas évalués

Pour terminer, ajouter le code PHP nécessaire pour réaliser le même type d'affichage en faisant varier cette fois-ci les balises HTML de <h5> à <h1> :

```
<?php
// variante avec insertion des parties variables dans les balises HTML
// moins lisible mais même résultat
for ($i = 5 ; $i >= 1; $i--)
{
    ?>
    <h<?php echo $i; ?>>Hello World !</h<?php echo $i; ?> >
<?php
}
?>
... et la suite de la page Web...
</body>
</html>
```

Tester sur différents navigateurs Web.

Attention : le contrôle visuel de ce qui est affiché par le navigateur ne suffit pas pour bien tester du code PHP ! Il est nécessaire de vérifier que le code HTML reçu est correct en relisant systématiquement le code source de la page Web reçue par le navigateur grâce au débogueur intégré.

3. DEUXIEME EXERCICE : VARIABLES D'ENVIRONNEMENT

On sait déjà que le navigateur Web dispose d'informations multiples sur l'ordinateur qui joue la page Web (voir l'objet JavaScript prédéfini Navigator). Ces informations (et beaucoup d'autres !) sont véhiculées (à l'insu de votre plein gré) par le protocole HTTP depuis le client Web jusqu'au serveur Web. Ainsi, un script PHP peut savoir quel type de navigateur a exprimé la demande de page de manière à générer du code HTML optimisé pour ce navigateur. De plus, d'autres informations concernant l'environnement serveur Web sont disponibles à travers des instructions PHP, ce qui permet éventuellement au script PHP de s'adapter au type de serveur Web qui est réellement utilisé (retenez que le serveur Web de développement n'est pas le serveur Web de publication et qu'un environnement PHP ne vaut pas l'autre !). Pour interroger les variables d'environnement côté serveur en PHP, le développeur dispose de l'ancienne instruction `getenv(XXX)` et du tableau superglobal prédéfini `$_SERVER`.

Voici le début de la page Web à générer :

En ce 11 September 2013, sur le serveur localhost, il est 8h 37mn.

Variable HTTP serveur (getenv())

Variable	Valeur
GATEWAY_INTERFACE	CGI/1.1
SERVER_NAME	localhost
SERVER_SOFTWARE	Apache/2.2.22 (Win32) PHP/5.3.13
SERVER_PROTOCOL	HTTP/1.1
REQUEST_METHOD	GET
QUERY_STRING	
DOCUMENT_ROOT	
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_CHARSET	
HTTP_ACCEPT_ENCODING	gzip,deflate,sdch
HTTP_ACCEPT_LANGUAGE	fr-FR;q=0.8,en-US;q=0.6,en;q=0.4
HTTP_CONNECTION	keep-alive
HTTP_HOST	localhost:8080
HTTP_REFERER	http://localhost:8080/TPPHP/
HTTP_USER_AGENT	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.66 Safari/537.36
REMOTE_ADDR	127.0.0.1
SCRIPT_FILENAME	C:/wamp/www/TPPHP/serveur.php
SERVER_ADMIN	admin@localhost
SERVER_PORT	8080
SERVER_SIGNATURE	

DATE DU JOUR

La date du jour peut être récupérée par l'instruction PHP `getdate()`. La donnée reçue est vue comme un tableau de variables, un « tableau associatif » PHP dans lequel sont stockées les différentes parties de la date, jour, mois, année...

Le début du script php pourrait donc ressembler à ceci :

```
<?php $cejour = getdate();?>
<html>
<head>
<title>Les variables HTTP</title>
</head>
<body>
<h2>En ce <?php echo $cejour["mday"] . " " . $cejour["month"] . " " . $cejour["year"]; ?>,
sur le serveur <?php echo $_SERVER["SERVER_NAME"]; ?>,
il est <?php echo $cejour["hours"] . "h " . $cejour["minutes"] . "mn" ;?>.<br />
```

Voir la documentation de référence
concernant les dates en PHP

Notez qu'il reste bien sûr possible de traduire en français le libellé du mois, soit par un algorithme de conversion en PHP, soit à l'aide d'instructions PHP spécifiques.

Par ailleurs, on utilise dans ce script le « *tableau associatif superglobal* » prédéfini `$_SERVER` pour récupérer le nom du serveur Web.

UTILISATION DE L'INSTRUCTION GETENV()

Cette instruction permet de récupérer un paramètre d'environnement quand on connaît son nom. Ici nous souhaitons afficher certains paramètres dans des lignes de tableau HTML, ce qui pourrait se réaliser en s'inspirant de cet extrait de script PHP :

```
<h3>Variable HTTP serveur (getenv())</h3>
<table border="1">
<tr>
<td><b>Variable</b></td>
<td><b>Valeur</b></td>
</tr>
<tr>
<td><?php echo "GATEWAY_INTERFACE" ; ?></td>
<td><?php echo getenv("GATEWAY_INTERFACE"); ?>&nbsp; </td>
</tr>
<tr>
<td><?php echo "SERVER_NAME" ; ?></td>
<td><?php echo getenv("SERVER_NAME"); ?>&nbsp; </td>
</tr>
<tr>
<td><?php echo "SERVER_SOFTWARE" ; ?></td>
<td><?php echo getenv("SERVER_SOFTWARE"); ?>&nbsp; </td>
</tr>
<tr>
<td><?php echo "SERVER_PROTOCOL" ; ?></td>
<td><?php echo getenv("SERVER_PROTOCOL"); ?>&nbsp; </td>
</tr>
```


PARCOURS DU TABLEAU ASSOCIATIF \$_SERVER

Pour la dernière partie de cette page Web dynamique, on voudrait obtenir les mêmes informations en utilisant le tableau prédéfini `$_SERVER` ; PHP offre souvent plusieurs manières d'aboutir au même résultat ce qui est assez perturbant pour le développeur débutant !

On peut aisément effectuer une boucle de parcours automatique de l'ensemble de ce tableau associatif à l'aide d'une boucle PHP `foreach(...)` qui permet de récupérer à la fois le libellé du poste du tableau ainsi que sa valeur :

```
<h3>Variable HTTP serveur ($_SERVER)</h3>
<table border="1">
<?php foreach ($_SERVER as $item=>$valeur ) { ?>
    <tr>
        <td><?php echo $item ; ?></td>
        <td><?php echo $valeur ;?>&nbsp; </td>
    </tr>
<?php } ?>
```

Syntaxe PHP particulière permettant de récupérer simultanément clé et valeur dans un tableau associatif PHP

A partir des informations et extraits de code fournis, réalisez la page Web dynamique permettant d'afficher la date, le nom du serveur Web et des informations d'environnement récupérées des 2 manières citées. Testez sur différents navigateurs et relevez les variantes (certaines informations concernent le serveur Web, d'autres, le navigateur).

4. TROISIEME EXERCICE : LES VARIABLES DE SESSION

Le protocole HTTP est dit « sans connexion persistante », c'est-à-dire que le serveur Web « oublie » le navigateur dès qu'il lui a servi une page. Comment dans ces conditions alimenter un « caddie » de page en page sur un site marchand ou encore s'assurer qu'un visiteur s'est bien authentifié avant d'afficher des informations confidentielles ? La solution communément utilisée repose sur l'usage *des variables de session* ; ce mécanisme est maintenant bien stabilisé et utilisable avec tout logiciel serveur Web et tout navigateur. N'oubliez pas comme toujours de vérifier le code HTML effectivement reçu par le navigateur.

Un navigateur est capable de fournir au serveur Web un « identifiant de session » caractérisant une « instance » particulière de ce navigateur ; si on ouvre simultanément ou successivement deux fenêtres d'Internet Explorer par exemple, chacune aura son identifiant de session unique. Cet identifiant de session est utilisé par le serveur Web pour allouer des ressources privées à chaque utilisateur du site, et ce, de manière transparente pour le développeur car *il lui suffit de déclarer qu'il utilise le mécanisme des variables de niveau session*.

En PHP, ces variables de niveau session sont stockées dans le tableau associatif *superglobal* `$_SESSION`. Mais comme ce mécanisme est « coûteux » pour le

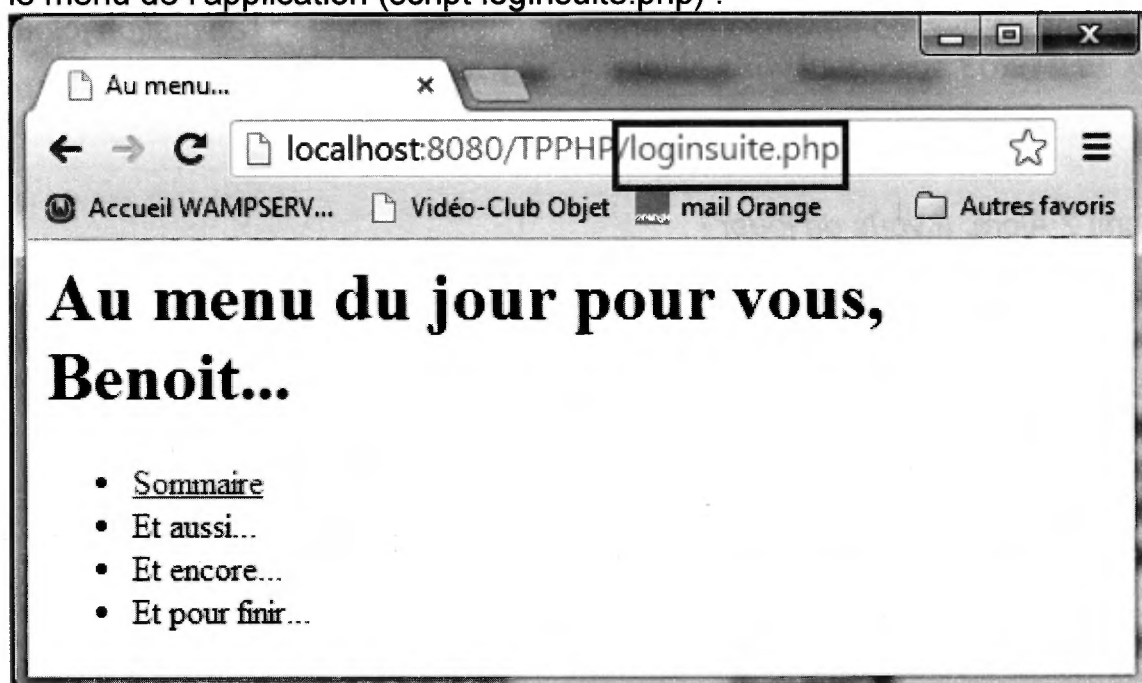
serveur Web, il est nécessaire en PHP de préciser explicitement que l'on veut utiliser les variables de session grâce à l'instruction `session_start()`.

Ne vous laissez pas abuser par le nom de cette instruction : elle *permet d'utiliser* le mécanisme des sessions et *non de démarrer* une session.

On va ici mettre en œuvre ce mécanisme à travers le dialogue de pages Web suivant :

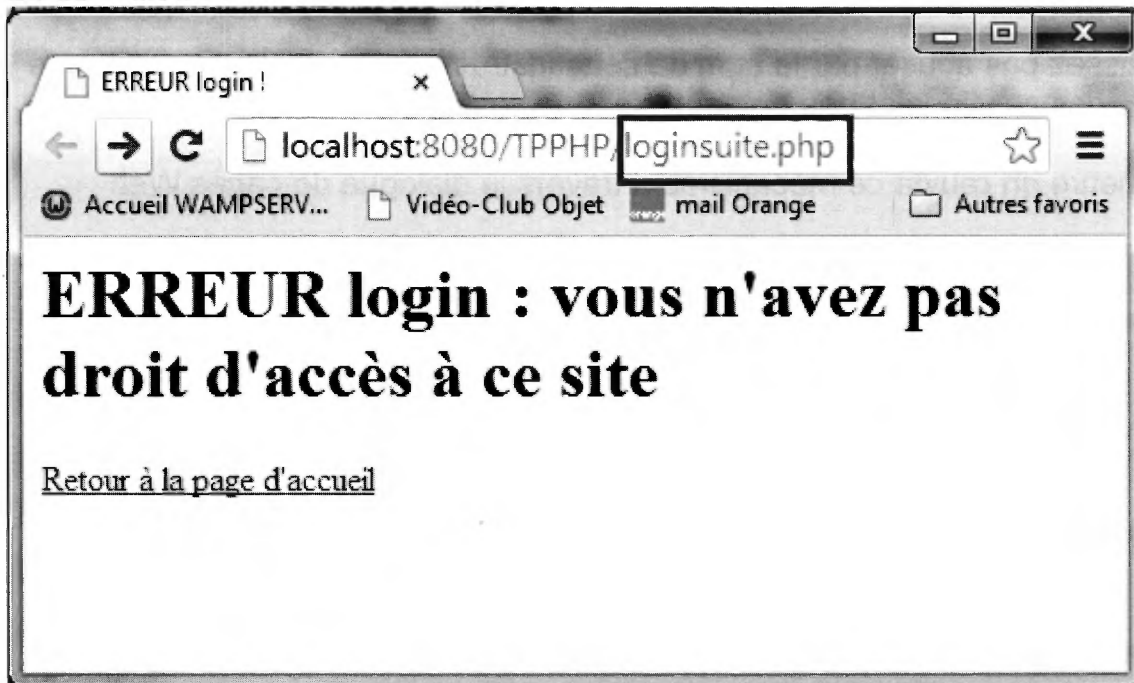


Dans ce form de login (login.htm), l'utilisateur saisit son nom (ici « Benoit ») et son mot de passe qui ne sont pas contrôlés (ce n'est pas encore l'objet de notre étude) ; ce formulaire déclenchera un script PHP (loginctrl.phpo) qui va simplement mémoriser le nom saisi, dans une variable de niveau session, puis se diriger vers le menu de l'application (script loginsuite.php) :



L'utilisateur peut maintenant mémoriser l'URL de cette page (copier/coller, favoris...) pour y retourner plus tard. Mais s'il s'agit d'une nouvelle session (autre instance du

navigateur, autre navigateur), le menu refusera de s'afficher et renverra au formulaire de connexion :



Notez bien que dans cet exemple, le *même script PHP* (loginsuite.php) fournira du *code HTML différent* affichant deux « *pages Web* » différentes selon les cas (ce qui est bien le but du PHP !), sous la *même URL* (adresse de page donc de script PHP).

Réalisation :

- Le *form* de login reste une simple page HTML basée sur un formulaire en méthode *POST* contenant un *input* type *text*, un autre *input* type *password* et les deux boutons classiques *reset* et *submit*. L'*action* associée au *form* sera le script PHP *loginctrl.php*.

Prenez le temps de bien comprendre cet enchaînement de pages ; l'animation jointe en annexe, *PHPExo3.ppsx* peut vous aider à vous y retrouver.

Bien compris ? Alors, à vous de jouer !

- Le script *loginctrl.php* doit récupérer le nom saisi dans le formulaire HTML posté, le mémoriser dans une variable de session à destination des pages futures, et renvoyer vers le script de menu *loginsuite.php*.
- PHP met à disposition les données saisies dans le formulaire précédent à travers les tableaux associatifs *\$_GET* et *\$_POST* selon la méthode d'envoi utilisée par le formulaire HTML, « *get* » ou « *post* ».
- Une variable de session est créée au moment de sa première affectation.
- On redirige vers une autre page/script grâce à l'instruction *header()* -qui permet de modifier/paramétrer bien des choses dans l'entête de message HTTP...-

Le script *loginctrl.php* pourrait donc être :

```
<?php
// page de controle de login
// page "aveugle" qui mémorise les var de ses
session_start();
$_SESSION["usrnom"] = $_POST["nom"];
header("location:loginsuite.php") ;
?>
```

La variable de session est créée lors de sa première affectation

Il s'agit d'un script « *aveugle* », 100% PHP, qui assure des traitements sans rien retourner au navigateur.

Petite précision sur le fonctionnement du protocole HTTP :

- Un *client Web* (navigateur) adresse en général au serveur Web une requête HTTP constituée d'une simple demande de page ; l'URL correspondante est complétée par des informations de service (adresses IP de l'expéditeur et du destinataire...) et adressée sur le réseau sous la forme d'un *entête de message HTTP* ;
- Dans le cas des *formulaires HTML*, les données saisies par l'utilisateur sont transmises au serveur soit dans l'*entête* de message (méthode « get »), soit dans le *corps* du message (méthode « post ») ; cette méthode « post » permet de véhiculer d'importants volumes de données mais elle est plus gourmandes en ressources réseau ;
- Les balises HTML `<meta http-equiv="..." />` permettent de modifier certaines informations de service contenues dans l'*entête* de message HTTP (mise en cache, indexation par moteurs de recherche...) ;
- Un *serveur Web* retourne la réponse sous forme d'un *entête de message HTTP* (informations de service) suivi du contenu HTML/JavaScript/CSS dans le *corps du message HTTP* ;
- L'instruction PHP `header()` permet de modifier certaines informations de service contenues dans l'*entête* du message HTTP retourné au client Web (date d'expiration de la page, mise en cache, redirection vers une autre URL...) ;
- En cas de redirection de la réponse HTTP -instruction PHP `header('location :...')`-, le développeur doit s'assurer que le script ne génère aucun contenu HTML/JavaScript/CSS à retourner au client (même pas un saut de ligne !) car il s'agit de se dérouter vers un autre script PHP ;
- Le code HTML/JavaScript/CSS résultant d'un script PHP sera effectivement envoyé au client Web dès la fin de l'évaluation du script ou lorsque l'interpréteur PHP rencontre l'instruction `exit()` ;

Dans notre cas, l'instruction `header('location :...')` est la dernière instruction rencontrée et la redirection vers le script suivant est donc immédiate mis c'est une bonne pratique de faire suivre `header('location:...')` par `exit()`.

L'affichage de la réponse sera assuré par le script *loginsuite.php*.

- Le script *loginsuite.php*, comme toute page à accès contrôlé du site, vérifie l'existence de la variable de session (instruction PHP *isset()*) ; si le test échoue, ce script affiche le message d'erreur sous forme d'une page HTML complète (qui peut encore se rediriger automatiquement vers le formulaire de login au bout de 3 secondes) ; si le test réussit, ce script affiche normalement le menu.
- Ce script commence par utiliser le mécanisme des variables de session :

```
<?php
// page de menu qui teste l'existence des var de session
// et affiche soit le menu soit un message d'erreur
session_start();
?>
```

- Le début de page HTML est variable au niveau du contenu de la balise HTML `<title>` :

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf_8">
<title>
<?php // titre page variable
if (!isset($_SESSION["usrnom"]))
{
    echo "ERREUR login !";
}
else
{
    echo "Au menu...";
}
?>
</title>
```

NB : en cas d'erreur, on peut aussi ajouter une instruction HTML `<meta http-equiv=... />` pour revenir à login.htm automatiquement après 3 secondes

Le corps de la page est bien entendu variable lui-aussi :

- en cas d'échec :

```
<body>
<?php // cas d'erreur ==> retour vers login.htm
if (!isset($_SESSION["usrnom"]))
{
    ?>
    <h1> ERREUR login : vous n'avez pas droit d'accès à ce site</h1>
    <p><a href="login.htm">Retour à la page d'accueil</a></p>
    <?php
    }
    else // authentification OK ==> menu
```

- et en cas de succès :

```

<?php
}
else // authentication OK ==> menu
{
?>
<h1>Au menu du jour pour vous, <?php echo $_SESSION["usrnom"]; ?>...</h1>
<ul>
    <li><a href="..">Sommaire</a></li>
    <li>Et aussi...</li>
    <li>Et encore...</li>
    <li>Et pour finir...</li>
</ul>
<?php
}
?>
</body>
</html>

```

Reproduisez ce script et testez en mémorisant le menu dans vos favoris pour y revenir avec la même instance du navigateur, quelques minutes plus tard, quelques heures plus tard, avec une autre instance de navigateur, avec un autre navigateur... Vous devez vérifier que votre session est bien associée à une instance d'un navigateur et que la session se détruit automatiquement après un temps d'inactivité (déterminé par le serveur Web).

NB : Variante de programmation du contrôle d'accès :

Avec cette construction, chaque page du site nécessitant un accès contrôlé a un contenu variable dans ses différentes parties de code HTML (<head>, <title>, <body>), ce qui rend le code assez confus et difficile à maintenir. On peut imaginer une autre construction où le script teste tout d'abord l'existence des variables de session pour rediriger immédiatement vers la page de login en cas d'erreur :

```

<?php
// controle variables de session
session_start();
if (!(isset($_SESSION['usrnom'])))
{
    header("Location: login.htm" ); // retour au form de login
    exit(); // c'est terminé !'
}
?>
<html>
<head>
    <title>Une page à accès sécurisé</title>

```

Ici, exit() est nécessaire pour rediriger immédiatement vers la page de login sans aller plus loin dans l'exécution de ce script

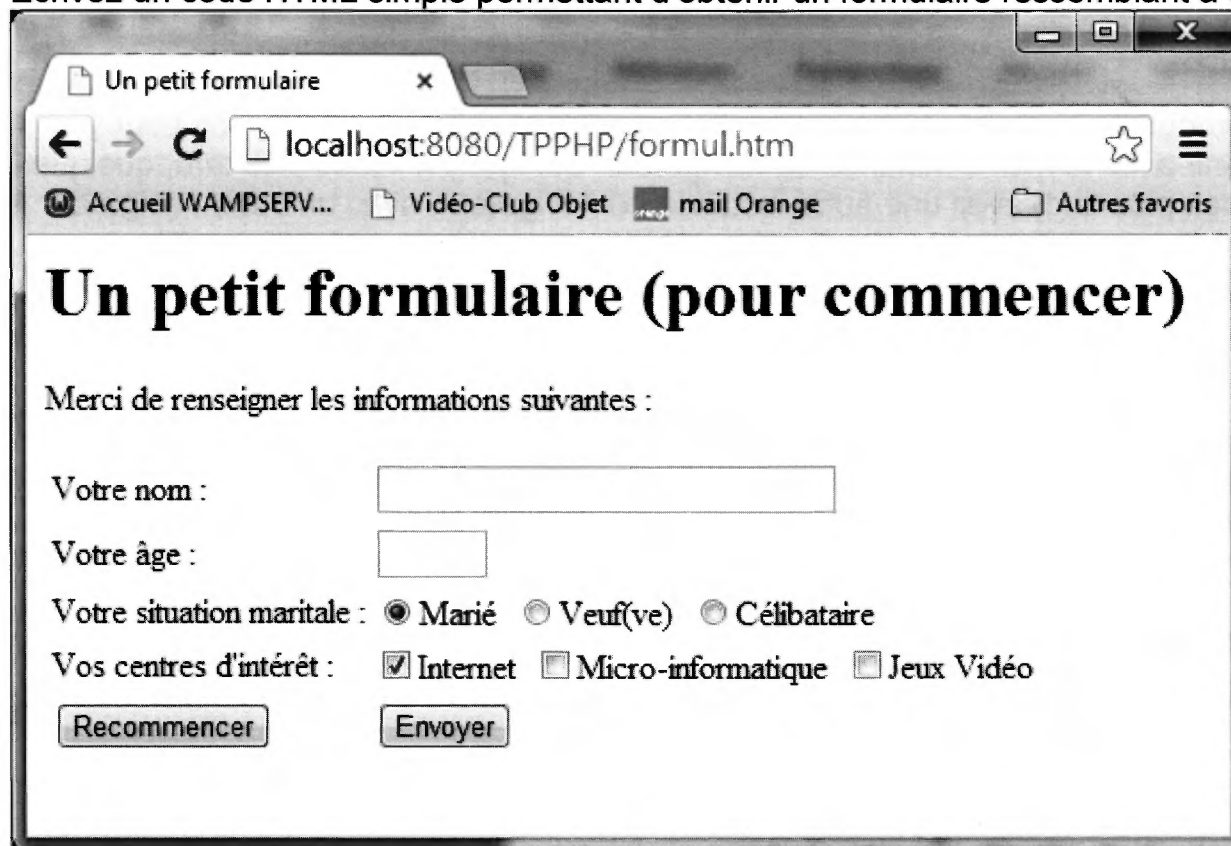
Cependant, avec cette construction, l'affichage d'un message d'erreur devient plus compliqué ; on y reviendra...

5. QUATRIEME EXERCICE : EXPLOITATION DES FORMULAIRES HTML

Dans cet exercice, il s'agit de se préparer à exploiter, côté serveur Web, en PHP, les données saisies dans un formulaire HTML. Si la transmission et la récupération des informations restent simples car il suffit de lire le contenu des tableaux associatifs *superglobaux* `$_GET` et `$_POST`, le traitement de ces données devient délicat dès lors qu'il s'agit de les lier à une base de données ; *la difficulté réside dans la préparation du libellé de la requête SQL à envoyer* au serveur de bases de données. Le but de cet exercice est donc de récupérer les informations saisies en formulaire HTML et de préparer une requête SQL, sans aller jusqu'à l'exécuter dans MySQL ; à l'issue de l'exercice, on prendra toute la mesure de l'aide apportée par un framework comme PDO pour accéder à une base de données...

FORMULAIRE HTML DE SAISIE

Ecrivez un code HTML simple permettant d'obtenir un formulaire ressemblant à ceci :



The screenshot shows a web browser window with the title 'Un petit formulaire'. The address bar shows 'localhost:8080/TPPHP/formul.htm'. The browser's bookmark bar includes 'Accueil WAMPSE...', 'Vidéo-Club Objet', 'mail Orange', and 'Autres favoris'. The form itself has the title 'Un petit formulaire (pour commencer)' and a message 'Merci de renseigner les informations suivantes :'. It contains the following fields and controls:

- 'Votre nom :' followed by a text input field.
- 'Votre âge :' followed by a text input field.
- 'Votre situation maritale :' followed by three radio buttons labeled 'Marié', 'Veuf(ve)', and 'Célibataire'.
- 'Vos centres d'intérêt :' followed by three checkboxes labeled 'Internet', 'Micro-informatique', and 'Jeux Vidéo'.
- At the bottom, there are two buttons: 'Recommencer' and 'Envoyer'.

Le *form* est bien entendu associé à un script PHP (par exemple *formul.php*) et il expédie ses données en méthode *GET* (uniquement à des fins de contrôle et de compréhension des échanges HTTP client/serveur).

Nommer tous les contrôles graphiques porteurs de données (par exemple *nom* et *age* pour les boîtes de texte, *marit* pour les boutons radio et *internet*, *micro*, *jeux* pour les cases à cocher) ; les boutons de commande, non porteurs d'information n'ont pas besoin d'être nommés en HTML (car tous les contrôles graphiques nommés en HTML sont envoyés au serveur Web).



SCRIPT FORMUL.PHP :

- Préparer des variables avant tout affichage ;
- Le nom est récupéré grâce à `$_GET["nom"]` ;
- L'âge est récupéré grâce à `$_GET["age"]` ;
- La situation maritale est reçue dans `$_GET["marit"]` comme une seule donnée dont la valeur correspond au *value* HTML du bouton sélectionné par l'utilisateur (vérifiez en observant la barre d'adresse quand vous soumettez le formulaire) ;

Voici donc le début du code du script nécessaire pour afficher la page de résultat :

```
<?php // analyse du formulaire reçu :
// init des var
$interet = ' ' ;// libelle des intérêts utilisateur
$marequ = 'insert into Matable values(' ;// partie constante de la requete sql

// recup nom et age
$marequ = $marequ . "'" . $_GET["nom"] . "'," . $_GET["age"] . "," ;

// recup situation maritale (bt radio dans le form)
$marequ = $marequ . "'" . $_GET["marit"] . "'," ;
```

NB : il s'agit bien de *constituer le libellé d'une requête SQL insert* où les valeurs sont séparées par des virgules et les chaînes de caractères entourées d'apostrophes.

NB : ici, on a utilisé la syntaxe PHP à base de *concaténations explicites* (opérateur PHP ' . '). Mais souvenez-vous qu'au premier exercice vous avez vu des variantes de syntaxe pour effectuer des *concaténations implicites* en PHP. Et il existe aussi un opérateur PHP d'auto-concaténation ' . ='.

On pourrait donc aussi bien écrire, pour concaténer le libellé de la requête :

```
// variante de syntaxe avec concaténations implicites
$marequ = 'insert into Matable values(' ;// partie constante de la
// récup nom et age
$marequ .= "'$_GET["nom"]',$_GET["age"]," ;
// recup situation maritale (bt radio dans le form)
$marequ .= "'$_GET["marit"]',," ;
```

Utilisez une syntaxe au l'autre selon ce qui vous semble le plus 'naturel' (ou logique) mais il est important de comprendre ces variantes de syntaxe pour pouvoir exploiter la prose disponible sur le PHP.

Et n'oubliez pas qu'une chaîne de caractères exprimée entre guillemets est plus coûteuse à traiter par l'interpréteur PHP car elle est évaluée caractère par caractère alors que la même chaîne exprimée entre apostrophes sera plus rapidement traitée ; réservez les guillemets aux chaînes contenant des caractères spéciaux et aux concaténations implicites.

- Pour les cases à cocher cela se complique car **seules les cases cochées par l'utilisateur seront envoyées au serveur Web** (vérifiez en observant la barre d'adresse quand vous soumettez le formulaire) ; le développeur ne peut donc pas savoir *a priori* s'il les recevra dans le tableau `$_GET` ! Il faut tester l'existence de la variable dans le tableau `$_GET` et, en cas de succès, récupérer sa valeur pour l'affichage ; il faudra aussi en déduire si on doit envoyer le chiffre 1 ou le chiffre 0 à la base de données. Cela peut s'écrire, pour les 2 premières cases :

```
// récup du/des centres intérêt utilisateur (checkbox dans le form)
// avec concaténation du libellé intérêts utilisateur
if(isset($_GET["internet"]))
{
    $marequ = $marequ . "1," ;
    $interet = "Internet," ;
}
else
{
    $marequ = $marequ . "0," ;
}

if(isset($_GET["micro"]))
{
    $marequ = $marequ . "1," ;
    $interet = $interet . " la micro-informatique," ;
}
else
{
    $marequ = $marequ . "0," ;
}
```

- Concernant l'affichage du résultat, tout est donc préparé par les traitements précédents ; il ne reste que la génération du code HTML :

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf_8">
  <title>Une petite réponse</title>
</head>
<body>
  <h2>Merci à vous, <?php echo $_GET["nom"] ; ?>.</h2>
  <p>Vous avez donc le bel âge de <b><?php echo $_GET["age"] ; ?></b> ans,
  vous êtes <b><?php echo $_GET["marit"] ; ?></b></p>
  <p>et vous vous intéressez à

  <b><?php echo $interet ; ?></b>.</p>

  <p>Je m'empresse d'envoyer la requête :<br />
  <b><?php echo $marequ ; ?><br /></b> à notre base de données.</p>
</body>
</html>
```

Reproduisez le code ci-dessus. Testez différentes valeurs dans le formulaire et observez bien les résultats obtenus ainsi que la barre d'adresse du navigateur. Il doit au moins rester à traiter le cas où l'utilisateur ne s'intéresse à rien...

VARIANTE POUR GERER LES CASES A COCHER :

Il est souvent utile de recevoir les cases cochées dans un tableau de variables PHP car on peut aisément parcourir un tableau grâce à l'instruction PHP *foreach(...)*. Pour arriver à cela, il est utile de nommer en HTML les cases comme un tableau PHP, comme ci-dessous :

```
<input type="checkbox" name = "interet[]" value="internet" checked>Internet&nbsp;
<input type="checkbox" name = "interet[]" value="micro">Micro-informatique&nbsp;
<input type="checkbox" name = "interet[]" value="jeux">Jeux Vidéo&nbsp;
```

Cet artifice n'a pas de sens particulier en HTML mais il force la récupération dans un tableau de variables PHP, des différentes cases cochées en formulaire. Dans le script PHP, il faut donc tester qu'au moins une case a été cochée puis parcourir tous les postes du tableau comme dans l'extrait de code suivant :

```

if(isset($_GET["interet"])) // au moins une case est cochée
{
    foreach ($_GET["interet"] as $item)
    {
        if ($item == "internet")
        {
            // case Internet a été cochée

        }
        else if ($item == "micro")
        {
            // case Micro-informatique a été cochée
        }
        else if (...)
    }
}

```

Il reste à traiter correctement la concaténation du libellé de la requête SQL, à déterminer s'il faut écrire des 1 ou des 0 dans la base de données... Il existe plusieurs logiques de solutions, vous en trouverez certainement une !

6. BILAN DES EXERCICES

FONDAMENTAUX

Un script PHP a pour vocation de générer/personnaliser du code HTML/JavaScript/CSS à retourner au navigateur de l'utilisateur.

On ne pourra donc jamais rien faire de mieux ou de plus avec PHP que ce que l'on saurait écrire en HTML/CSS/JavaScript.

Tout le code HTML/CSS/JavaScript *constant* peut s'écrire « naturellement », en dehors des balises `< ?php ... ?>`, au sein du script PHP.

Dès que du code HTML/CSS/JavaScript est *variable*, qu'il doit être adapté/généré côté serveur, il est nécessaire de programmer la logique du traitement correspondant en langage PHP dans une balise spécifique `<?php ... ?>`.

SYNTAXE PHP

Terminer chaque instruction par un signe point-virgule.

Les variables ne sont pas déclarées et leur type s'auto-adapte au contenu courant (ce qui n'est pas une raison pour programmer n'importe comment...).

Les tableaux de variables PHP peuvent être manipulés classiquement par indices numériques mais PHP propose aussi des *tableaux associatifs* dans lesquels les indices sont avantageusement remplacés par des libellés bien plus clairs.

PHP met à disposition du développeur des tableaux associatifs prédéfinis (« superglobaux ») comme `$_SERVER`, `$_GET`, `$_POST`, `$_SESSION`.

VARIABLES DE SESSION

Un serveur Web « oublie » un client Web dès qu'il lui a servi la page demandée ; pour garder la trace des différentes navigations d'un utilisateur (sur un même site), il est possible d'utiliser des variables de session qui persistent au-delà de la durée de vie du script PHP sur le serveur.

Une session correspond à une instance d'un navigateur et a une durée de vie limitée.

EXPLOITATION DES FORMULAIRES COTE SERVEUR

Tous les champs de saisie *nommés* d'un formulaire HTML sont adressés au serveur Web par le navigateur Web lors de la soumission du formulaire (clic sur bouton submit ou encore méthode `.submit()` du formulaire invoquée par JavaScript).

Dans un même groupe de *boutons radio*, celui qui est coché donne sa valeur à la variable reçue en PHP.

Pour les *cases à cocher*, seules les cases cochées par l'utilisateur seront transmises au serveur ; le script PHP devra donc *tester leur existence* avant de pouvoir les exploiter.

Etablissement référent

Centre Afpa Nice

Equipe de conception

Benoit Hézard

Remerciements :

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.
« toute représentation ou reproduction intégrale ou partielle faite sans le
consentement de l'auteur ou de ses ayants droits ou ayants cause est
illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction
par un art ou un procédé quelconques. »

Date de mise à jour 10/03/2016
afpa © Date de dépôt légal mois année

