TP PHP n°2

I. Truck, Université Paris 8, 2019–2020

1 Variables superglobales : les *cookies*

a. Comme on l'a vu dans le TP précédent, les sessions permettent donc de stocker des informations pendant toute la durée d'une connexion, et d'échanger ces informations d'une page à l'autre. Mais si l'on veut sauvegarder les informations, même lorsque l'internaute n'est plus connecté ? ⇒ il s'agit alors de *cookies*.

b. Un *cookie* est un petit fichier qui est stocké **sur le disque dur de l'internaute** et non pas sur le serveur.

c. En principe, on doit sauvegarder dans un *cookie* **une seule information**. Par exemple, pour un forum, si l'on veut enregistrer le pseudonyme de l'internaute, son adresse mail, sa date d'inscription et son *design* (thème) d'affichage préféré, il vaut mieux utiliser quatre *cookies*.

d. C'est le navigateur du poste client qui gère l'emplacement de stockage des *cookies* (on n'a donc pas besoin de connaître cet emplacement). Par exemple,
   - avec Google Chrome, ils se trouvent dans le répertoire [Profil Utilisateur]\Local Settings\Application Data \Google\Chrome\User Data\Default\Cache ;
   - Internet Explorer enregistre chaque *cookie* dans un fichier différent ;
   - Edge utilise plusieurs répertoires (lire cette page `https://stackoverflow.com/questions/31945227/edge-browser-cookie-store-location-and-access` pour plus d'informations)
   - Mozilla Firefox enregistre tous ses *cookies* dans un seul fichier ;
   - Safari enregistre tous ses *cookies* dans un seul fichier d'extension `.plist` ;
   - Opera enregistre tous ses *cookies* dans un seul fichier et le chiffre.

e. Chaque *cookie* a un **nom** et une **valeur**, et possède une *date d'expiration*. Quand la date expire, il est automatiquement effacé du disque dur du client. Un *cookie* contenant une adresse mail correspondra par exemple au couple `('adrMail','isis.truck@univ-paris8.fr')`.

f. On utilise la fonction `setcookie` pour **créer** un *cookie*. Elle nécessite au minimum 3 paramètres : le nom, la valeur et la date d'expiration, exprimée en secondes écoulées depuis le 1er janvier 1970. On obtient la date d'expiration en appelant la fonction `time()` et en lui ajoutant un temps en secondes. Par exemple, pour créer le *cookie* pour le thème d'affichage préféré, on écrira :
`setcookie('themePrefere', 'fleursBleues', time() + 3600*24*7);`
Combien de temps le *cookie* sera-t-il valide ?

g. Il existe d'autres paramètres à la fonction `setcookie()`, notamment le 7e et dernier qui correspond à l'option `httpOnly`. Pour protéger le *cookie*, il faut mettre ce paramètre à `true`. En effet, ce paramètre prend pour valeur un booléen qui définit si le *cookie* sera accessible uniquement par le protocole HTTP. Cela signifie que le *cookie* ne sera pas accessible *via* des langages de scripts comme Javascript. Cette configuration permet de limiter les attaques via XSS (bien qu'elle ne soit pas supportée par tous les navigateurs). *Cf.* explications ici :
`https://fr.wikipedia.org/w/index.php?title=Cookie_(informatique)&oldid=144238927#Vol_de_cookie`

h. Mettre les autres paramètres à `false` ou `null` (consulter la signature de la méthode `setcookie()` pour plus de détails). Attention, `setcookie` crée un (seul) *cookie* et s'écrit **avant tout code HTML**.

i. On **lit** le contenu d'un *cookie* en regardant le contenu du tableau associatif `$_COOKIE[]`. Pour afficher ce que contient le *cookie* de nom `adrMail`, il suffit de taper `echo $_COOKIE['adrMail']`. Comme pour les sessions, il faut d'abord vérifier que le *cookie* existe bel et bien avec `isset()`.

j. On **met à jour** le contenu d'un *cookie* en utilisant la fonction `setcookie()`. Affecter une valeur avec = à la variable `$_COOKIE[]` n'aura **aucun impact** car `$_COOKIE[]` reste une **variable locale**. Si l'on change sa valeur, cela n'aura de portée que dans le script courant et cela ne modifiera pas le cookie. Seule la fonction `setcookie()` modifie véritablement `$_COOKIE`. Par ailleurs, juste après un `setcookie()` (dans le même script), la variable `$_COOKIE` n'est pas encore renseignée. C'est parce que

`$_COOKIE` est mise à jour quand la page est chargée à cause de la nature "sans état" (*stateless*) du protocole HTTP, c'est-à-dire que chaque requête est indépendante. En d'autres termes, **le cookie n'est mis à jour que lorsque la réponse est renvoyée au client**, il faut donc attendre la prochaine requête du client pour que `$_COOKIE` soit positionnée. Si l'on veut absolument connaître la valeur du cookie tout de suite, il suffit bien sûr d'écrire : `setcookie('nomCookie', 'valeur', time()+3600); $_COOKIE['nomCookie'] = 'valeur';`

k. Pour finir, le souci qui peut se poser en refaisant un `setcookie()` (pour une mise à jour du cookie) est de ne pas connaître la durée du cookie déclaré préalablement. Si cette durée est connue dans le(s) script(s) PHP, pas de problème, sinon, il est préférable de créer un cookie contenant la durée du ou des cookie(s) déclaré(s).

l. On peut voir, sous Chrome par exemple, l'état des cookies en appuyant sur `Ctrl-Shift-J` pour ouvrir la fenêtre *Developer Tools*, onglet *Resources*.

2 Exercice avec les *cookies*

a. Ne pas oublier de **créer un nouveau répertoire**, dédié à ce TP n°2.

b. Reprendre les fichiers `index.php`, `entete.php`, `piedPage.php` et `corps.php` en les dupliquant.

c. Dans `corps.php`, ajouter le traitement des données saisies dans le formulaire (`nom, prenom, phrase`) par des *cookies*.

d. Afficher ensuite le résultat des *cookies* dans une autre page PHP (nommée `cookies.php` et qui sera appelée depuis `corps.php` *via* un bouton) pour vérifier le bon fonctionnement.

e. Enfin, modifier dans le fichier `cookies.php` la valeur du *cookie* `prenom` en mettant `Emilie` à la place. Tester que la modification a bien été prise en compte.

3 Manipuler les fichiers

— Si l'on souhaite sauvegarder des données de taille moyenne et ce, longtemps (car les variables sont éphémères et les *cookies* réduits à une seule valeur), on peut vouloir stocker des informations dans des fichiers.

— En principe, on range les fichiers dans un répertoire *ad hoc* qui doit être en mode `execute` pour les *others* (`o`, *cf.* `chmod`). Si problème, mettre les droits en `rwx` sur le répertoire. Les fichiers doivent, quant à eux, être en mode `r` et `w` pour les *others*.

— Avant toute chose, il faut *ouvrir* le fichier. On utilise la fonction `fopen(nomFichier, mode)` qui renvoie un objet de type *flux*. `nomFichier` doit être mis entre *quotes* et `mode` peut être égal à (non exhaustif) :
  — `r` : cela permet d'ouvrir le fichier en lecture seule, et de placer le pointeur de fichier au début du fichier ;
  — `r+` : cela permet d'ouvrir le fichier en lecture et écriture, et de placer le pointeur de fichier au début du fichier ;
  — `w` : cela permet d'ouvrir le fichier en écriture seule, de placer le pointeur de fichier au début du fichier et de réduire la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer ;
  — `w+` : cela permet d'ouvrir le fichier en lecture et écriture, de placer le pointeur de fichier au début du fichier et de réduire la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer ;
  — `a` : cela permet d'ouvrir le fichier en écriture seule, de placer le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer ;
  — `a+` : cela permet d'ouvrir le fichier en lecture et écriture, de placer le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.

  On écrit donc, par exemple l'instruction `$flux = fopen('fic.txt', 'r+');`.
  Nota bene : Les systèmes d'exploitation utilisent différents caractères pour les nouvelles lignes. Lors de l'écriture d'un fichier texte et l'insertion d'une nouvelle ligne, il faut utiliser le bon caractère. Les systèmes Unix utilisent `\n` comme nouvelle ligne, les systèmes Windows utilisent `\r\n`, et les systèmes Macintosh utilisent `\r`. Pour palier ce problème, on peut utiliser l'option `'b'` pour forcer le fichier à être écrit en mode binaire, sans traduction des données. Il suffit d'ajouter `'b'` comme dernier caractère du paramètre `mode`. Lire le manuel de `fopen` pour plus de détails.

— Pour *fermer* le fichier, il faut, en fait, fermer le flux. On utilise la fonction `fclose(nomFlux)`, donc on écrira par exemple `fclose($flux);`

— Pour lire le contenu du fichier, caractère par caractère, on utilise la fonction `fgetc()`.

— Pour lire le contenu du fichier, ligne par ligne, on utilise la fonction `fgets()` qui renvoie une ligne (stockée dans une `string`). Exemple : `$ligne = fgets($flux);`. Pour lire 10 lignes, il faut faire une boucle. La fin se détecte avec `feof($flux);` qui renvoie un booléen.
— Pour écrire dans un fichier, il faut utiliser `fputs()` qui prend en paramètre le flux et le texte à écrire. Exemple : `fputs($flux, 'Texte à écrire');`
— ***Nota bene* important** : PHP (qui est un langage côté serveur) peut accéder aux ressources **du serveur** et plus particulièrement au **système de fichiers du serveur**. Ainsi, tout ce qui est écrit plus haut s'applique sur les fichiers stockés et créés sur le serveur (hormis, bien sûr, les cookies, qui, eux, sont stockés sur le disque dur du client). Donc, les fichiers sur `handiman`, par exemple. C'est bien eux qu'on manipule, et non pas les fichiers stockés du côté du client. Il existe tout de même deux exceptions à la règle disant que PHP ne peut que manipuler les fichiers côté serveur : l'*upload* de fichiers et le *download* de fichiers. Mais pour ces deux dernières "manipulations", on remarque que "l'initiative" est laissée au client (l'utilisateur), et non au serveur. (source : `http://web.doc.free.fr/php6.php`)

4 Exercice avec les fichiers

a. Le but est d'écrire dans un fichier les noms et prénoms entrés dans le formulaire (*cf.* TP n°1, exercice 2). Copier d'abord les fichiers correspondant au formulaire dans le répertoire du TP2 et modifier le code en fonction des questions ci-dessous.
b. Reprendre le fichier `corps.php` — en faisant une copie `corpsFic.php` — et modifier le traitement associé au formulaire : il s'agit donc de sauvegarder chaque couple (`nom, prenom`), à raison d'un couple par ligne et en insérant une virgule entre le nom et le prénom, dans un fichier nommé `nomInternautes.txt` à enregistrer dans le même répertoire que celui dans lequel on travaille.
c. Après le traitement, vérifier que le fichier se remplit correctement en créant une page nommée `nomInternautes.php` qui affiche la liste des internautes, ligne par ligne.
d. Reprendre le fichier `corpsFic.php` — en faisant une copie `corpsFic2.php`— et modifier la façon de sauvegarder chaque couple (`nom, prenom`) dans le fichier `nomInternautes.txt` : il faut à chaque fois **écraser** l'ancien couple. Pour faire cela, il faut que le pointeur de fichier soit systématiquement placé en début de fichier. Or, après un `fputs()`, le pointeur est situé juste après le dernier caractère écrit. Il faut donc repositionner le pointeur avec `fseek()`. Par exemple, `fseek($flux,0)` permet de repositionner le pointeur au début du fichier.

5 Question subsidiaire : les captcha

a. Il est demandé de modifier le formulaire de saisie du nom-prénom, de sorte de forcer le remplissage d'un **captcha** avant de valider la saisie du nom-prénom. Il y a plusieurs façons de faire un *captcha*. Lire, par exemple, ces explications : `http://www.finalclap.com/faq/46-php-creer-captcha`.
b. Ajouter un captcha au formulaire en utilisant une des méthodes possibles. Penser à également à celle-ci : en ajoutant simplement un *radio button* et un champ image qui dit "décocher cette case", le robot ne saura pas lire l'image, donc ne décochera pas le bouton radio.
c. Est-ce que le captcha créé est **accessible** ? Comment faire un captcha accessible ? Rendez-le accessible.