











# Les cookies

# Sommaire

<b>Principe de fonctionnement des cookies</b> .....	3
A. Création d'un cookie.....	3
B. Afficher un cookie.....	4
C. Supprimer un cookie.....	6
D. Exemple d'utilisation des cookies.....	7
E. Intégration des cookies dans le mini-site.....	8

Crédits des illustrations :  
© DR.

## Les repères de lecture

 Retour au chapitre	 Définition	 Objectif(s)	 Espace Élèves	 Vidéo / Audio
 Point important / À retenir	 Remarque(s)	 Pour aller plus loin	 Normes et lois	 Quiz

# Principe de fonctionnement des cookies

À l'instar des sessions, les cookies permettent l'enregistrement d'informations et la propagation de celles-ci d'une page à une autre, mais au lieu de le stocker dans un fichier sur le serveur, les informations sont stockées dans un fichier sur le client, c'est-à-dire sur le navigateur.

Le mécanisme des cookies en PHP permet la création, la modification et la suppression du fichier contenant les informations recueillies ainsi que la lecture des informations contenues dans ce fichier.

L'intérêt des cookies repose sur le fait qu'ils peuvent être exploités longtemps après que l'utilisateur ait quitté le site. Ils ont en effet une durée de vie paramétrable qui peut aller jusqu'à plusieurs mois. C'est la raison qui fait que le système des cookies est utilisé pour identifier et suivre les visiteurs notamment dans le cas de besoins de type statistique, dont les questions sont généralement : combien de personnes consultent le site et est-ce que ces personnes sont déjà venues ?

Si lors de la première visite, nous envoyons un cookie à l'utilisateur, nous sommes donc capables s'il revient, de vérifier si ce cookie existe et donc de voir qu'il s'agit d'une deuxième visite. L'intérêt réside donc dans la persistance de l'information, ce que les sessions ne peuvent permettre.

Notons que nous pouvons envoyer des cookies avec d'autres langages de développement que PHP, comme JavaScript, par exemple.



Les cookies font partie des en-têtes HTTP, ce qui impose que `setcookie()` soit appelée avant tout affichage de texte.

## A. Création d'un cookie

Pour créer un cookie, nous utilisons simplement la fonction `setcookie()`. Cette fonction possède les arguments suivants dans cet ordre précis :

- l'argument *name* : donne un nom au cookie (de la même manière que nous créons un nom pour une variable) ;
- l'argument *value* : définit la valeur du cookie (de la même manière que nous affectons une valeur à une variable). Cette valeur doit être une chaîne de caractères ou un nombre ;
- l'argument *expire* : définit en secondes la durée de vie du cookie (par défaut, le cookie a une durée de vie de 0 seconde et disparaît donc immédiatement après sa création) ;
- l'argument *path* : définit le chemin (path) sur le serveur pour lequel le cookie sera disponible. Par exemple, le cookie est-il valable pour tous les répertoires du site ou bien seulement pour un répertoire précis. Cet argument est facultatif, la valeur par défaut est le répertoire courant où le cookie a été défini. Si la valeur est `'/'`, le cookie sera disponible sur l'ensemble du site. Si la valeur est `'/dir/'`, le cookie sera uniquement disponible dans le répertoire `/dir/` et les éventuels répertoires enfants ;

- l'argument *domain* : définit le domaine pour lequel le cookie est disponible. Facultatif.

**Par exemple :** si nous possédons le domaine *mondomaine.com* nous pouvons le spécifier ;

- l'argument *secure* : définit le niveau de sécurité du protocole de la transaction HTTP : soit 0 pour http soit 1 pour https. Ceci est également facultatif, et la valeur par défaut est 0.

Voici donc la création d'un cookie (sans les arguments facultatifs) au moyen de la fonction *setcookie()* :

```
setcookie('nom_du_cookie', 'valeur du cookie', time()+3600);
```

Fig.1

Les arguments *name* et *value* ne posent pas de problème particulier. Attardons-nous en revanche sur l'argument *expire*. Nous constatons que nous attribuons la valeur « *time()+3 600* ». L'usage de la fonction *time()* que nous connaissons, permet de dire à PHP que la durée de vie du cookie débute à l'instant précis où il est envoyé. Nous additionnons 3 600 à *time()*, ce qui va ainsi définir que la durée de vie du cookie est de 3 600 secondes, soit 1 heure.

Si nous voulons que le cookie ait une durée de vie de 2 heures, c'est-à-dire 2 fois 3 600 secondes, nous écrivons :

```
time() + 7200
```

Fig.2

Si nous voulons que le cookie ait une durée de vie de 3 jours, c'est à dire 3 fois 24 heures, nous écrivons :

```
time() + 259200
```

Fig.3

Nous pouvons ainsi exprimer une durée de vie du cookie sur plusieurs jours, semaines ou mois.

Ajoutons que comme pour les sessions ou la fonction *header*, les cookies sont envoyés au navigateur dans les entêtes HTTP et donc aucun affichage ne doit être effectué avant l'implantation de la fonction *setcookie()*.

## B. Afficher un cookie

L'accès aux informations contenues dans les cookies s'effectue avec la variable superglobale *\$\_COOKIE[]* qui est prédéfinie et dont la syntaxe à base de crochets indique qu'il s'agit d'un tableau de données.

Si nous voulons afficher le cookie précédemment créé et qui a pour nom « nom\_du\_cookie », nous passons simplement entre crochet le nom donné au cookie dans l'argument name et nous écrivons :

```
echo $_COOKIE['nom_du_cookie']; // affiche valeur du cookie
```

Fig.4

Mais attention car les cookies ne seront accessibles qu'au rechargement de la page courante.

En effet le mécanisme est effectué en deux étapes :

Nous créons un fichier de test que nous appelons cookie.php qui contient la fonction setcookie() et un echo pour afficher la valeur du cookie.

Lorsque nous ouvrons pour la première fois dans le navigateur la page cookie.php, alors PHP envoie le cookie au navigateur et comme celui-ci vient d'être envoyé il n'est pas encore disponible. Si nous voulons l'afficher avec echo, nous obtenons une erreur de type « Notice: Undefined index: nom\_du\_cookie ». Le cookie n'est pas encore disponible et donc n'est pas encore défini.

Si nous rechargeons la page (touche F5) ou si nous y revenons plus tard, alors le cookie est maintenant disponible et nous pouvons l'afficher. Il est important de bien connaître ce mécanisme pour éviter des erreurs lors de nos développements. Les cookies ne sont pas accessibles immédiatement après leur création.

Au passage, nous serons amenés à tester de plus en plus de fonctions PHP et il est recommandé de créer sous Wamp un répertoire dédié à ces tests dans lesquels nous créons nos fichiers de test. Ce répertoire peut par exemple s'appeler test\_php, et les fichiers de tests dans lesquels nous pourrions écrire des commentaires seront naturellement nommés en fonction du type de test que nous menons (cookie.php, session.php, etc). Nous pourrions ainsi y revenir lorsque le besoin s'en fera sentir pour se rappeler un point qui nous a paru important concernant telle ou telle mise en œuvre.

Puisque les cookies sont des tableaux de données, nous pouvons également utiliser un print\_r pour les consulter en mode développement.

```
print_r($_COOKIE);
```

Fig.5

Résultat:

```
Array
(
    [nom_du_cookie] => valeur du cookie
)
```

Fig.6

Il est naturellement possible de transmettre plusieurs cookies en même temps en utilisant plusieurs fois la fonction `setcookie()`.

#### Exemple:

```
setcookie('premier_cookie', 'valeur 1', time()+3600);
setcookie('second_cookie', 'valeur 2', time()+3600);
Array
(
    [premier_cookie] => valeur 1
    [second_cookie] => valeur 2
)
```

Fig.7

Pour obtenir si besoin une structuration des valeurs transmises, nous pouvons également utiliser le même nom pour plusieurs cookies en utilisant les crochets et créer ainsi un tableau de données en créant `nom_du_cookie[1]` et `nom_du_cookie[2]`.

```
setcookie('nom_du_cookie[1]', 'valeur 1', time()+3600);
setcookie('nom_du_cookie[2]', 'valeur 2', time()+3600);
```

Fig.8

Nous récupérons ainsi les deux valeurs des cookies dans un tableau de données:

```
Array
(
    [nom_du_cookie] => Array
        (
            [1] => valeur 1
            [2] => valeur 2
        )
)
```

Fig.9

Nous pouvons tirer utilement partie de cette structuration, par exemple avec des boucles.

L'accès à chaque cookie s'effectuant naturellement dans ce cas avec la syntaxe:

```
echo $_COOKIE['nom_du_cookie'][1];
```

Fig.10

## C. Supprimer un cookie

Pour supprimer un cookie il suffit d'utiliser la fonction `setcookie()` pour recréer un cookie avec le même nom mais un argument *value* vide.

```
setcookie('nom_du_cookie', ''); // supprime le cookie nom_du_cookie
```

Fig.11

## D. Exemple d'utilisation des cookies

Nous créons un script simple dans un fichier compteur.php qui va proposer d'afficher à l'internaute la date de sa dernière visite sur la page ainsi que le nombre de visite qu'il a effectuée. Nous testons donc l'existence ou non des deux cookies que nous utilisons: \$\_COOKIE['date\_visite'] contient l'heure de visite et \$\_COOKIE['nombre\_visite'] contient le nombre de visite.

Pour cet exemple, nous indiquons une durée de vie d'une semaine.

```
<?php
$heure_visite = date('d m Y h:i:s');
setcookie('date_visite', $heure_visite, time()+604800);
if (isset($_COOKIE['date_visite']))
{
    if (isset($_COOKIE['nombre_visite']))
    {
        $nombre_visite = $_COOKIE['nombre_visite'] ;
        $nombre_visite = $nombre_visite + 1 ;
        $message = 'Bonjour, vous avez visité cette page '. $nombre_
visite .' fois';
    }
    else
    {
        $nombre_visite = 1 ;
        $message = 'Bonjour, vous avez déjà visité cette page ' .
$nombre_visite .' fois';
    }
    setcookie('nombre_visite', $nombre_visite, time()+604800);
    $message .= '<br>la date de votre dernière visite est le ' . $_
COOKIE['date_visite'] ;
}
else
{
    $message = 'Bonjour,<br> C\'est votre première visite sur
cette page';
}
?>
<?php if(isset($message)): ?>
<p><?php echo $message ?></p>
<?php endif?>
```

Fig. 12

Le résultat de ce script varie en fonction du nombre de visites que nous simulerons en rechargeant la page (F5).

Première visite :

```
Bonjour,
C'est votre première visite sur cette page
```

Fig. 13

Deuxième visite :

```
Bonjour, vous avez déjà visité cette page 1 fois  
la date de votre dernière visite est le 28 11 2012 14:50:13
```

Fig.14

Troisième visite :

```
Bonjour, vous avez déjà visité cette page 2 fois  
la date de votre dernière visite est le 28 11 2012 15:50:18
```

Fig.15

## E. Intégration des cookies dans le mini-site

Nous pouvons optimiser notre mini-site des villes en intégrant un système de statistiques basé sur les cookies et la base de données.

Un cookie sera transmis au client et nous pourrons ainsi enregistrer dans la base le nombre de visites de l'internaute.

Le fichier que nous appellerons `statistique.php` contiendra : l'envoi du cookie nommé `visite` et l'enregistrement dans la base de la valeur du cookie. Afin d'identifier de manière unique l'internaute, nous lui attribuons un identifiant unique généré aléatoirement par la fonction `uniqid()` qui retourne un identifiant unique, sous la forme d'une chaîne de 13 caractères.

```
echo uniqid(); // génère par exemple la chaîne 50c1fc1f9a567
```

Fig.16

Nous souhaitons donc enregistrer deux informations dans le cookie.

- l'identifiant unique généré par `uniqid()`;
- le nombre de visites.

Il est pertinent d'envisager de créer un tableau de données pour enregistrer ces deux informations données, mais la valeur d'un cookie doit être une chaîne de caractères ou un nombre et ne peut pas contenir de tableaux de données.

Nous allons donc utiliser la très pratique fonction PHP `serialize()` qui transforme les tableaux de données en chaîne de caractères en préservant la structure du tableau.



## Les fonctions serialize / unserialize

Soit un tableau de données \$tab contenant deux valeurs et dont la structure est la suivante :

```
Array
(
    [0] => a
    [1] => b
)
```

Fig. 17

Appliquons la fonction serialize sur ce tableau :

```
$var = serialize($tab);
echo $var;
```

Fig. 18

Le résultat de la serialisation est la chaîne de caractères suivantes :

```
a:2:{i:0;s:1:"a";i:1;s:1:"b";}
```

Fig. 19

Nous pouvons récupérer le tableau avec unserialize :

```
$tab2 = unserialize($var);
```

Fig. 20

Nous avons bien récupéré notre tableau initial

```
Array
(
    [0] => a
    [1] => b
)
```

Fig. 21

Puisque nous savons transformer un tableau en chaînes de caractères, il nous est alors possible de l'enregistrer dans cookie. Notre tableau que nous appellerons 'user\_stat' contiendra donc les deux informations que nous souhaitons enregistrer dans le cookie.

- l'identifiant unique généré par uniqid : \$user\_stat['web\_user\_id'] ;
- le nombre de visites : \$user\_stat['web\_user\_visit'].

Sur la base de l'exemple de script précédent, nous effectuons les conditions permettant de vérifier la présence ou non du cookie visite. Si celui-ci n'existe pas nous créons les valeurs à insérer dans le cookie et nous l'insérons avec INSERT INTO dans la base de données. Si celui-ci existe, nous modifions les valeurs du cookie et nous mettons à jour avec UPDATE la base de données.

Voici le code final :

```
<?php
if (isset($_COOKIE['visite'])) // le cookie existe
{
    // valeur du cookie
    $cookie_value = $_COOKIE['visite'];
    $cookie_value = unserialize($cookie_value);
    $web_user_id = $cookie_value['web_user_id'];
    $web_user_visit = $cookie_value['web_user_visit'];
    // mise à jour du nombre de visites
    $cookie_value['web_user_visit'] ++;
    // serialisation pour enregistrer les données dans le cookie
    $stat_data = serialize($cookie_value);
    // mise à jour dans la base du nombre de visite pour cet internaute
    $mysqli->query('UPDATE stat SET web_user_visit = '. $web_user_visit .'
    WHERE web_user_id = "' . $web_user_id .'"');
}
else // le cookie n'existe pas
{
    $web_user_id = uniqid();
    $nombre_visite = 1;
    $user_stat['web_user_id'] = $web_user_id;
    $user_stat['web_user_visit'] = $nombre_visite;
    // serialisation pour enregistrer les données dans le cookie
    $stat_data = serialize($user_stat);
    // ajout dans la base du nombre de visite pour cet internaute
    $mysqli->query('INSERT INTO stat (web_user_id, web_user_visit)
    VALUES ("'. $web_user_id .'", '. $nombre_visite .')' );
}
// envoi du cookie
setcookie('visite', $stat_data, time()+259200);
?>
<hr>
<?php
/*code temporaire destiné au développement pour vérifier le contenu du cookie
et effectuer les tests de nouvelle visite.
Le fichier delete_cookie.php supprime le cookie
et redirige vers la page courante avec la variable server HTTP_REFERER */
print_r($_COOKIE);
?>
<br>
<a href="delete_cookie.php">suppression du cookie</a>
```

Fig.22

Nous renommons ensuite le fichier en inc\_stat.php et l'incluons dans les fichiers index.php et ville.php. Afin que les entêtes HTML ne génèrent pas d'erreur en cas d'affichage de contenu avant l'envoi du cookie, l'inclusion est effectuée en dessous du fichier dédié à la connexion à la base de données comme suit :

```
<?php require('inc_connexion.php'); ?>
<?php require('inc_stat.php') ?>
```

Fig.23

Il ne reste plus qu'à créer une page de consultation des statistiques que nous pourrions consulter dans l'administration. Pour la structure du code, ce fichier sera développé sur le modèle de ajout.php.

Aucune difficulté particulière n'est prévue. Afin d'obtenir l'ordre décroissant du nombre de visites, nous ajouterons la commande ORDER BY dans la requête MySQL.

```
SELECT stat_id, web_user_id, web_user_visit FROM stat ORDER BY web_user_visit DESC'
```

Fig.24

Cette commande demande à MySQL de trier le résultat par l'attribut (ou champ) web\_user\_visit.

Par défaut, cette commande trie les résultats selon un ordre croissant. Afin de l'obtenir dans l'ordre décroissant, nous ajoutons la commande DESC. Notons que la commande ASC permettrait un tri par ordre croissant.

Nous écrivons également deux requêtes complémentaires qui vont demander à MySQL de nous retourner le nombre total de visites et le nombre total de visiteurs.

Notons que nous utilisons le mot clef AS suivi d'une valeur (AS total\_visites et AS total\_visiteurs) qui peuvent ainsi être récupérés dans le tableau \$row et nous permettent de rendre plus clair notre code.

Résultat:

```
// calcul par Mysql du nombre total de visites
$result = $mysqli->query('SELECT SUM(web_user_visit) AS total_visites FROM stat');
$row = $result->fetch_array();
$total_visites = $row['total_visites']; //
$result->free();
// calcul par Mysql du nombre total de visiteurs
$result = $mysqli->query('SELECT COUNT(web_user_id) AS total_visiteurs FROM stat');
$row = $result->fetch_array();
$total_visiteurs = $row['total_visiteurs']; //
$result->free();
```

Fig. 25 Page de consultation des statistiques dans l'administration

Nous sommes désormais capables de pister (tracker) l'internaute durant sa navigation sur notre site. Nous avons récupéré dans la base le nombre global de visites du site. Il serait très intéressant d'optimiser ce code afin d'obtenir un résultat par page afin de déterminer quelles sont les pages les plus visitées.

Les valeurs utilisées dans les statistiques peuvent être nombreuses: adresse IP, type de navigateur, page de provenance de l'internaute. Ce sont des valeurs anonymes. Mais il est également possible de proposer à l'internaute qu'il s'identifie pour bénéficier par exemple de service personnalisé. Dans ce cas, les informations recueillies peuvent être recoupées et traitées par type de critères personnalisés: âge, sexe, métier, etc. qui peuvent nous permettre d'améliorer grandement le site web en fonction de la cible du site.

Notons enfin que la législation impose d'indiquer clairement à l'internaute que des informations personnelles le concernant sont recueillies et qu'il dispose d'un droit d'accès et de modification à ces informations.











[esecad.com](http://esecad.com)