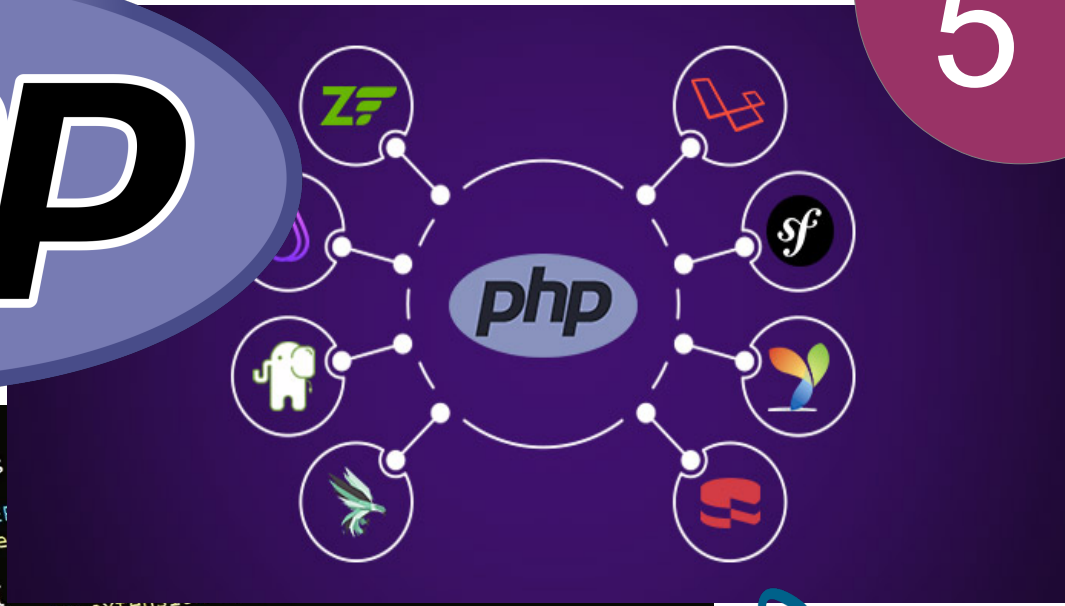
php

5




```
 * @var boolean
 */
define('PSI_INTERNAL_XML', false);

if (version_compare("5.2", PHP_VE
    die("PHP 5.2 or greater is re

}
if (!extension_loaded("pcre")) {
    die("phpSysInfo requires the pcre extens
        properly.");

}

require_once APP_ROOT.'/includes/autoloader.inc.php';

// Load configuration
require_once APP_ROOT.'/config.php';

if (!defined('PSI_CONFIG_FILE') || !defined('PSI_DEBUG')) {
    $tpl = new Template("/templates/html/error_config.html");
    echo $tpl->fetch();
    die();
```

MySQL®

Messmer Lucie

Dans ce cours :

- Sessions
- Fonctions Date
- Regexp

Exercice PHP4

Les Sessions

Dans la famille des variables dites ''superglobales'', nous avons vu que $_GET et $_POST nous permettaient de transmettre des données d'une page vers une autre, mais sitôt que nous quittions cette dernière, les informations disparaissaient.

Afin d'ameliorer la persistance des données, nous avons également vu qu'il était possible de les stocker dans des fichiers. Cependant cette operation est gourmande en ressource et un peu lourde la plupart du temp au regard de la nature des informations que l'on souhaite conserver durant notre navigation.

C'est pour résoudre ce problème que la superglobale $_SESSION a été créé.

Elle permet de conserver les informations serveur durant le temps où l'utilisateur est sur la page.

Les sessions vont nous permettre de conserver des variables sur toutes les pages d'un site.

Ouvrir une session

Syntaxe

```
session_start();
```

Cette fonction doit ABSOLUMENT être appelée avant tout envoi au navigateur et donc avant même le <!DOCTYPE>, sans quoi vous ne pourrez utiliser les variables de sessions. Si la session n'existe pas déjà, elle est créée ainsi qu'un numero d'ID unique qui lui est associée (''ID de session''). Dans tous les cas il nous faut déclarer cette fonction si l'on veut manipuler nos variables.

Fermer une session

Syntaxe

```
session_destroy()
```

Cette fonction supprime la session. L'ID et toutes les variables que nous y avons stockés sont détruits. On crée généralement un bouton ou un lien ''déconnexion'' pour lancer cette opération.

Cependant, la plupart du temps l'utilisateur quitte le site sans se ''déconnecter'' et le serveur ne peut le savoir. Il existe donc un timeout qui permet de fermer automatiquement une session.

NB : *C'est généralement l'inactivité qui est timée. Si aucune page n'est demandée ou rafraichie pendant une laps de temps défini, la déconnexion s'opère.*

La variable de session, $_SESSION, est un tableau. La seule difference avec les autres tableaux que nous avons utilisé, est que celui-ci ne s'efface pas à la fin d'un script.

Déclaration

Syntaxe

```
$_SESSION['ma_variable'] = valeur;
```

exemple :

```
// ouverture de session session_start();
// création et affectation de 2 informations à stocker
$_SESSION['prenom']= 'Jasper';

$_SESSION['role'] =  'Développeur';
```

Consultation

```
// ouverture de session session_start();
// récuperation des 2 variables précédemment stockée

$prenom = $_SESSION['prenom'];

$role = $_SESSION['role'];
// affichage
echo "Bonjour " . $prenom . ", vous êtes " . $role;
```

Suppression

`session_unset()` detruit toutes les variables de la session courante.

Utilisez unset() pour détruire une variable de session

exemple : `unset($_SESSION['role']);`

Autres fonctions

Les dates

Gérer les dates en PHP est relativement simple grâce à toute une collection de fonctions dediées.
Nous allons voir ici les fonctions les plus utilisés liées aux dates.
Avant d'entrer dans le vif du sujet, un petit mot sur ce qu'on appelle le timestamp : c'est un nombre entier qui correspond au nombre de secondes écoulées depuis le 1 janvier 1970.
Toujours êxtrèmement utilisé, il commence pourtant à être délaissé par les développeurs : sa taille qui ne cesse de grossir, ses limitations et le fait que la gestion des dates devient de plus en plus importante et complexe font que d'autres solutions se mettent en place.

La fonction time()

Elle permet de générer le timestamp de la date courante.

exemple :

```
echo time() ;
// va afficher 1664291867
```

La fonction date()

Une des fonctions les plus utilisées, elle permet de convertir un timestamp dans un format de date de notre choix.
date(format, timestamp)

Voici un tableau avec quelques valeurs possibles pour le format.

Caractère	Description
j	Jour du mois sur 2 chiffres sans zéro initial (1 a 31)
d	Jour du mois sur 2 chiffres avec zéro initial en fonction du jour (01 a 31)
w	Jour de la semaine au format numérique (0 pour dimanche, 6 pour samedi)
z	Jour de l'année (0 à 366)
W	Numéro de semaine dans l'année (les semaines commencent le lundi)
n	Mois sur 2 chiffres sans zéro initial (1 à 12)
m	Mois sur 2 chiffres avec avec zéro initial en fonction du mois (01 à 12)
y	Année sur 2 chiffres
Y	Année sur 4 chiffres
G	Heure au format 24h sans zéro initial (0 à 23)
H	Heure au format 24h avec zéro initial en fonction de l'heure (00 à 23)
I	Minutes (00 à 59)
s	Secondes (00 à 59)

Pour obtenir la date actuelle ainsi que l'heure, voici un exemple :

```
// récupération du timestamp
      $time = time();
// affichage
      echo date("j.n.Y G:i ",$time);
```

La fonction mktime()

Si je veux obtenir des informations sur une autre date, il me faut connaitre son timestamp pour l'utiliser avec la fonction date().
Heureusement, nous allons pouvoir trouver le timestamp de n'importe quelle date grace à mktime().

`mktime(heure, minute, sec, mois, jour, année)`

Il est en effet très courant d'utiliser ces deux fonctions conjointement pour obtenir des informations utiles comme le jour d'une date precise.

Ici par exemple, nous pouvons savoir quel jour était le 13 décembre 1997 :

```
// récupération du timestamp
$time = mktime(0,0,0,12,13,1997);
// affichage
echo date("w",$time);
// affiche 6 : 6e jour de la semaine, ce qui correspond à un samedi.
```

Avec la fonction date() certains caractères permettent de récupérer des informations littérales, mais par defaut elles seront en anglais.

Modifions l'exemple ci-dessus :

```
// récupération du timestamp
$time = mktime(0,0,0,12,13,1997);
// affichage
 echo date("l",$time);
// nous allons avoir Saturday en reponse
```

Une fonction va nous permettre d'améliorer cela.

La fonction strftime()

C'est l'équivalent de la fonction date(), mais elle gère toutes les langues. Son utilisation demande donc un peu plus de ressources.

`strftime(format, timestamp)`

Avant de l'utiliser, il va nous falloir préciser la langue que nous souhaitons utiliser. Pour ça, on va utiliser la fonction **setlocale()**.

`setlocale(constante, valeur)`

Pour du français :

```
setlocale(LC_ALL, 'fr_FR@euro', 'fr_FR', 'fr_fr')
```

Et un tableau de valeurs possibles pour le format de strftime() page suivante.

%A – Nom complet du jour de la semaine
%a – Nom abrégé du jour de la semaine
%d – Jour du mois sur 2 chiffres avec zéro initial en fonction du jour (01 a 31)
%w – Jour de la semaine au format numerique (0 pour dimanche, 6 pour samedi)
%j – Jour de l'année (0 a 366)
%V – Numéro de semaine dans l'année (les semaines commencent le lundi)
%m – Mois sur 2 chiffres sans zero initial (1 a 12)
%B – Nom complet du mois
%b – Nom abrege du mois
%y – Année sur 2 chiffres
%Y – Année sur 4 chiffres
%H – Heure au format 24h avec zéro initial en fonction de l'heure (00 a 23)
%M – Minutes (00 à 59)
%S – Secondes (00 à 59)

La fonction checkdate()

Autre fonction bien pratique qui permet de verifier la validité d'une date. Elle renvoie true ou false.

```
checkdate(mois, jour, année)
```

NB : Comme pour la fonction mktime(), le mois est déclaré avant.

Cette fonction peut être déconcertante car elle interprètera certaines date mauvaises comme bonnes.
Si vous mettez 33 pour un jour d'un mois de 31 jours, elle décalera de façon logique au mois suivant de 2 jours et répondra donc true. La fonction mktime() a le même comportement.

Les expressions regulieres

Les expressions régulières (communément appelées regexp) sont des modèles crées à l'aide de caractères ASCII permettant de manipuler des chaînes de caractères, c'est-a-dire permettant de trouver les portions de la chaîne correspondant au modèle.
Pour faire simple, il s'agit d'un systeme fort ingenieux et très puissant permettant de retrouver un mot, une phrase ou plus généralement un motif dans un texte.
Les expressions régulières permettent de rechercher des occurrences (c'est-a-dire une suite de caractères correspondant à ce que l'on recherche) grâce à une série de caractères spéciaux.

Les symboles ^ et $

Ils indiquent respectivement le début et la fin d'une chaîne de caractère et permettent donc de la délimiter.
^debut correspond a toute chaîne qui commence par debut
fin$ correspond a toute chaîne qui se termine par fin
^chaine$ correspond a une chaîne qui commence et se termine par chaine

Les symboles *, +, et ?

Ils indiquent respectivement "zero ou plusieurs", "un ou plusieurs" et "un ou aucun". Ils permettent de donner une notion de nombre.
abc+ : string qui contient ab suivie de un ou plusieurs c (abc, abcc...)
abc* : string qui contient ab suivie de zero ou plusieurs c (ab, abcc...)
abc? : string qui contient ab suivie de zero ou un c (ab ou abc)
^abc+ : string commençant par ab suivie de un ou plusieurs c (abc, abccc...)

Les accolades {x, y}

Permettent de donner des limites de nombre plus complexes.
abc{2} : string qui contient ab suivie de 2 c (abcc)
abc{2,} : string qui contient ab suivie de 2 c ou plus (abcc, abccccccc...)
abc{2,4} : string qui contient ab suivie de 2, 3 ou 4 c (abcc, abccc ou abcccc)

NB : Le premier nombre de la limite est obligatoire, pas le dernier qui, s'il est omis, signifie qu'il n'y a pas de limite maximale.

Les parenthèses ()

Permettent de grouper une séquence de caractères.
a(bc)* : string qui contient a suivie de 0 ou plus bc (a, abcbc...)

La barre verticale |

Se comporte comme l'operateur OU.
un|le chien : string qui contient un chien ou le chien
(a|b)* : string qui contient une suite de a ou de b(a, aaabaabba...)

Le point .

Remplace n'importe quel caractère, une fois.
^.{3}$: string qui contient exactement 3 caractères quelconques (a6J, A12...)

Les crochets []

Permettent de définir des ensembles de caractères.
[aei] : string qui contient un a, un e ou un i
Le signe - place entre deux caractères permet de definit un intervalle.
[a-z] : string qui contient un caractère compris entre a et z

Le caractère ^ après le premier crochet indique la négation de l'ensemble.
^[^a-zA-Z] : string qui ne commence pas par une lettre.
En dehors des [] pour rechercher un caractère faisant partie des caractères spéciaux, il suffit de le faire précéder d'un \ (antislash).
Dans les crochets, chaque caractère représente ce qu'il est. Pour representer un] il faut le mettre en premier (ou après un a si c'est une exclusion).
[\+?{}.] : string qui contient un des 6 caracteres \ ,+, ?, { , } ou .
[]-] : string qui contient] ou -

Les éléments PCRE

PCRE est l'acronyme de ''Perl Compatible Regular Expressions''. Avec ce type de fonction, nous allons gagner en lisibilité et en simplicité.

\b indique une limite de mot dans une string
\B indique ce qui n'est pas une limite de mot
\d est équivalent à [0-9] (digit)
\D est équivalent à [^0-9] (non digit)
\s indique un espace blanc \t, \r, \n, \f. (space)
\S indique ce qui n'est pas un espace blanc \t, \r, \n, \f.
\w indique un mot qui correspond a la classe [0-9a-zA-Z_]
\W indique ce qui n'est pas un mot, soit [A0-9a-zA-Z_]

Fonctions utilisant les REGEXP

Les fonctions PHP modernes utilisant les expressions régulières sont issues du langage PERL.
En voici quelques unes.

`preg_match('/regexp/', chaine, result)`

Cette fonction permet d'évaluer le texte passé en argument (string) grâce au motif, c'est-à-dire l'expression régulière (regexp) que l'on insère entre les slashs (on peut egalement utiliser des #).
La fonction renvoie true s'il y a correspondance entre la string et le motif, false dans le cas contraire.
Il est possible de stocker toutes les occurences dans un tableau passé en dernier argument (result).

exemple :

L'expression régulière de ce premier exemple exige que le premier caractère de la string soit absolument une lettre majuscule suivie de 'onjour'.

Modifions la string afin de constater l'efficacité :

```
$string = 'Bonjour à tous';
$test = preg_match('/^[A-Z]onjour/', $string, $result);
echo $test;
// va afficher :    1
var_dump($result);
// va afficher : array(1) {    [0]=> string(7) "Bonjour" }
$string = 'bonjour à tous';
$test = preg_match('/^[A-Z]onjour/', $chaine, $result);
echo $test;
// va afficher :    0
var_dump($result);
// va afficher : array(0) {    }
```

preg_replace('/regexp/', $rempla, string)

Cette fonction permet d'évaluer le texte passé en argument (string) grâce au motif (regexp).
La fonction remplace toutes les occurences trouvées par $rempla.

exemple :

```
$chaine = 'Bonjour à tous';
$rempla = 'Au revoir';
$result = preg_replace('/[A-Z]onjour/', $rempla, $chaine);
echo $result;
// va afficher : Au revoir à tous
$chaine = 'J\'ai 3 livres de chevet';
$rempla = 'plusieurs';
$result = preg_replace('/[2-9]+/', $rempla, $chaine);
echo $result;
// va afficher : J'ai plusieurs livres de chevet
```

preg_split('/regexp/', chaîne)

Cette fonction permet d'eclater la chaîne passee en argument (chaîne) grace au motif (regexp). Un tableau est genéré en retour contenant les sous-chaînes qui auront été séparées par le motif.

exemple :

```
print_r(tableau) permet de visualiser le contenu d'un tableau avec moins d'information que
var_dump().
$chaine = 'PHP mySQL, Javascript, jQuery';
$tableau = preg_split('/[\s,]+/', $chaine);
print r($tableau);
// Va afficher : Array ([0] => PHP [1] => mySQL [2] => Javascript [3] => jQuery )
```

Fonction filter_var

Cette petite fonction bien pratique va nous faciliter la tâche pour certaines verifications specifiques et nous éviter le casse-tête que peut être la rédaction d'une expression régulière correcte.

`filter_var(chaine, filtre, option)`

Le texte passé en argument (chaine) se voit appliquer un filtre particulier.
En cas de succés, les données sont renvoyées filtrées, dans le cas contraire, c'est false qui est renvoyé.
Il existe des filtres de ''validation'' et des filtres de ''nettoyage''.
Les premiers ne transforment pas la chaîne testée mais la valide ou non.
Les seconds sont capables d'éventuellement la modifier afin de la faire correspondre à un schéma précis.

Voyons deux exemples parmis les plus courants :

exemple :

Verifier la validité d'une adresse email

```
$email = 'prenom.nom@email.com';
var_dump(filter_var($email, FILTER_VALIDATE_EMAIL);
// Va m'afficher l'email car il est correct, dans le cas contraire, affiche false
```

Supprimer tous les caractères non numériques

```
$login = 'random456random21';
var_dump(filter_var($login, FILTER_SANITIZE_NUMBER_INT);
// Va m'afficher "45621"
```