PHP

Introduction:

- Afficher une page différente en fonction de l'utilisateur, de l'environnement, ...

- Utiliser un langage de programmation évolué, par exemple PHP.

Introduction: C'est quoi PHP ?

PHP (Personal Home Page / Hypertext Preprocessor) est un langage de programmation de scripts côté serveur permettant de produire des pages web dynamiques.

- Langage récent (crée en 1994)
- Langage de script
- Support d'orienté objet depuis la version 5
- Présence d'un interpréteur côté serveur
- Typage dynamique
- Intégré au code HTML
- Syntaxe proche du C et du Java
- Interface simple avec beaucoup de SGBD

Introduction: Modèle d'exécution

1) Le client (navigateur) demande une page PHP
2) Le serveur web exécute le code de la page
   - Lancement de l'interpréteur
   - Exécution du code
3) Le serveur web renvoie le résultat (HTML) de l'exécution
4) Le client affiche le résultat

Remarque:

Pour le client, il est impossible de voir le code PHP. Seul le résultat de l'exécution est récupéré par le client.

Introduction: Créer des scripts PHP

- Les scripts PHP :
  - Sont de simples fichiers ”texte” (extension conseillée .php) à créer avec un éditeur de texte.
  - Contiennent du code PHP mélangeables à du code HTML.
  - Sont exécutés côté serveur par un “interpréteur” php (parser php).
- PHP s’intègre dans l’HTML entre `<?php ?>`
- Les instructions se finissent par ;
- Les commentaires sont soit entre /* et */, soit après // ou #

Exercice d'application: première page PHP

1. Lancer XAMP Control Panel, puis démarrer le serveur Apache.
2. Dans le dossier htdocs (C:\xampp\htdocs) de XAMP, créer un nouveau dossier 'application'
3. Dans le dossier application créer une page index.php puis insérer le code suivant.

```
<!-- index.php -->
<html>
<head> <title>Ma première page PHP</title>
</head>
<body>
   <?php echo '<h1>Nous sommes le ' . date("j/m/Y") . "</h1>";?>
</body>
</html>
```

4. Lancer le navigateur puis entrer le lien suivant 'localhost/application/index.php'

les entrées/sortie:

- Entrées: A l'aide de formulaires / argument passés au script via terminal
- Sorties: Pages Web/ terminal

On peut afficher avec la commande echo (avec ou sans parenthèses) print est équivalente à echo.

Les variables :

- Les variables sont préfixées par $
- Leur nom suit les règles classiques
  Exemple : `$my_var_01`
- Les noms sont sensibles à la casse : $var != $Var
- Pas de déclaration, typage implicite
  Exemples :

```
$my_var_01 = 54 ; // Maintenant, c'est un entier
$my_var_01 = "Big Data" ; // Maintenant, c'est une chaine
```

Les types :

- Entiers : 54
- Flottants : 54.3
- Chaînes : ”54” ou ’54’
- Booléens : false ou true
- Tableaux

Remarques:

- `isset($var)` : renvoie true si $var existe
- `unset($var)` : détruit $var
- `is_integer($var)`, `is_string($var)`,... : renvoie true si $var est un entier, une chaîne,...

Les constantes:

- On définit les constantes à l'aide de la commande define
  Exemple : `define("PI", 3.14)`
- On les utilise directement (sans $) : `echo PI`
- Test d'existence : `defined("PI")` renvoie 1, `defined("Pi")` renvoie 0

Les instructions conditionnelles:

If else	Switch case

`if ( cond ) {`
`...`
`}`
`elseif ( cond ) {`
`...`
`}`
`else {`
`...`
`}`
	`switch ( expr ) {`
	`case VALEUR1 :`
	`...`
	`break ;`
	`case VALEUR2 :`
	`...`
	`break ;`
	`default :`
	`...`
	`break ;`
	`}`

Les boucles:

```
for ( init ; cond ; modif ) {
...
}
```

```
while ( cond ) {
...
}
```

```
$abc = ['a', 'b', 'c'];
foreach($abc as $cle => $valeur)
{
echo "$valeur";
}
```

```
do {
...
} while ( cond ) ;
```

Exercice d’application:

1. Ecrire un script PHP (parfait.php) qui détermine si un entier N est parfait ou non. Un entier est dit parfait s'il est égal à la somme de ses diviseurs. Exemple 6 = 3 + 2 +1.

Les tableaux:

- Chaque élément du tableau a une clé et une valeur
- Pas de déclaration du tableau
- Les valeurs des éléments ne sont pas forcément toutes du même type

Exemple 1:

```
$tab[0] = 20;
$tab[1] = "mohammed";
$tab["LI"] = false;
```

Exemple 2:

```
$tab = array(20,"mohammed", "LI" => true);
```

Les tableaux: Parcours d'un tableau

- Parcours avec la boucle for
- Parcours spécifique:

```
foreach( $tab as $valeur ) {
echo $valeur;
}
```

```
foreach( $tab as $cle => $valeur ) {
echo $cle.':'.$valeur;
}
```

Les tableaux: Fonctions prédéfinies

- count($tab) : compte le nombre d'éléments initialisés
- current($tab) : retourne la valeur de l'élément en cours
- key($tab) : retourne l'indice de l'élément en cours
- reset($tab) : déplace le pointeur vers le premier élément
- next($tab) : déplace le pointeur vers l'élément suivant
- prev($tab) : déplace le pointeur vers l'élément précédent

Les tableaux: Fonctions prédéfinies

- sort($tab) : trie les valeurs et réaffecte les indices
- asort($tab) : trie les valeurs et ne réaffecte pas les indices
- rsort($tab) : équivalant à sort mais dans l'ordre inverse
- arsort($tab) : équivalant à asort mais dans l'ordre inverse
- ksort($tab) : trie les indices
- krsort($tab) : équivalant à ksort mais dans l'ordre inverse
- usort($tab,$critere), uasort($tab,$critere), uksort($tab,$critere) : trie selon un critère

Exercice d'application:

1. Dans un script tab.php déclarer le tableau suivant:

```php
$tab = array('age' => '20', "mohamed", "test" => true);
```

2. Afficher le tableau en utilisant la fonction print_r. Quel est l'indice de la valeur 'mohamed' ?
3. Afficher les pairs (clé, valeur) du tableau en utilisant la boucle foreach.

Les variables prédéfinies:

PHP fournit un très grand nombre de variables prédéfinies accessibles à tous les scripts.

Les variables Superglobales sont des variables internes qui sont toujours disponibles, quel que soit le contexte :

- $_SERVER : Variables de serveur et d'exécution
- $_GET : Variables HTTP GET
- $_POST : Variables HTTP POST
- $_FILES : Variable de téléchargement de fichier via HTTP
- $_REQUEST : Variables de requête HTTP
- $_SESSION : Variables de session
- $_ENV : Variables d'environnement
- $_COOKIE : Cookies HTTP

Les variables prédéfinies:

Le tableau associatif $GLOBALS contenant les références sur toutes les variables globales actuellement définies dans le contexte d'exécution global du script.

- $php_errormsg (le dernier message d'erreur)
- $http_response_header (en-têtes de réponse HTTP)
- $argc et $argv (le nombre et le tableau d'arguments passés au script)

Les constantes prédéfinies:

Le langage PHP met à disposition du programmeur des constantes propres au script qui ne peuvent pas être redéfinies. Les constantes prédéfinies du PHP sont :

- __FILE__ : nom du fichier en cours
- __LINE__ : numéro de ligne en cours
- PHP_VERSION : version de PHP
- PHP_OS : nom du système d'exploitation qui est utilisé par la machine qui fait tourner le PHP.

Exercice d'application:

1. Modifier l'exercice parfait.php pour qu'il vérifier si un nombre passé comme argument au script est un nombre parfait (utiliser le tableau des arguments $argv).

2. Modifier l'exercice pour qu'il vérifier le nombre passé en URL (utiliser le tableau superglobale $_GET)

Exemple:

- URL : index.php?num=5
- echo $_GET['num'];

Les chaînes de caractères:

- Délimites par ' : contenu non interprété
- Délimitées par " : contenu interprété
- Les unes peuvent contenir les autres
- Concaténation avec .

Exemple:

```
$nom = "bigdata"; // Contient "bigdata"
$nom1 = "filière $nom"; // Contient "filière bigdata"
$nom2 = 'pas filière $nom'; // Contient "pas filière $nom"
$nom2 = 'filière ' . $nom; // Contient "filière bigdata "
```

Remarque:

Accès à un caractère : `$nom[0];`

`strlen($nom);` : longueur de `$nom`

Comparaison avec ==, === ou `strcmp($nom1,$nom2);`

Les chaînes de caractères: Fonctions prédéfinies

- str_repeat(ch, nb) : répétition
- strtolower(ch) : minuscules
- strtoupper(ch) : majuscules
- ucwords(ch) : initiales en majuscules
- ucfirst(ch) : 1re lettre en majuscule
- ltrim(ch, liste) : suppression de caractères au début
- rtrim(ch, liste) : suppression de caractères à la fin
- trim(ch, liste) : suppression de caractères au début et à la fin

Les chaînes de caractères: Fonctions prédéfinies

- strstr(ch, ch2) : recherche sensible à la casse (retourne tous les caractères de ch depuis la 1re occurrence de ch2 jusqu'à la fin)
- stristr(ch, ch2) : recherche insensible à la casse
- substr(ch, indice, N) : extraction de chaîne de caractères
- substr_count(ch, ssch) : décompte du nombre d'occurrences d'une sous-chaîne
- str_replace(oldssch, newssch, ch) : remplacement
- strpos(ch, ssch) : position
- explode($sep, $ch, $limit) : divise une chaîne en un tableau en utilisant un séparateur

Exercice d'application:

Extrayez le nom de serveur de l'email suivant: samir.kamal@bigdata.com:

- Méthode 1: Utiliser les fonctions: substr, strpos, et strlen.
- Méthode 2: Utiliser la fonction explode.

Les fonctions:

- Pas de type pour les paramètres ou la valeur de retour
- Le nom est insensible à la casse
- Le résultat est renvoyé avec la commande return
- Une seule valeur de retour
- Passage des paramètres par valeur (par défaut) et par référence : &$param

Exemple:

```
function fonc1($arg_1, &$arg_2, $arg_3 = "BigData")
    {
        // $arg_1 => passage par valeur
        // &$arg_2 => passage par référence
        // $arg_3="BigData" => valeur par défaut
        return $valeur;
    }
```

Les fonctions:

Les variables utilisées à l'intérieur d'une fonctions sont détruites à la fin, sauf :

- si on les définit avec static
- si on les définit avec global

Remarque:

Comme d'autres langages de programmation, le langage PHP support les fonctions **anonymes** et les fonction **fléchées**.

Exercice d'application:

1. Ecrire une fonction qui affiche la table de multiplication d'un nombre entré comme paramètre.

2. Ecrire une fonction fléchée qui renvoie le carré d'un nombre (la fonction prend un nombre comme paramètre).

POO en PHP:

La programmation orientée objet (POO) est un modèle de programmation informatique qui met en œuvre une conception basée sur les objets. Ce type de programmation a fait son apparition dans la version 3 de PHP.

Exemple de classe en PHP:

```
class Product
{ //Commence par une Majuscule
    public $title; //Propriétés
    function getTitle()
    { //Méthodes
    }
}
$p = new Product();
$p->getTitle();
echo $p->title; // On pointe vers la propriété (sans le '$')
```

POO en PHP: Visibilité

La visibilité d'une propriété ou d'une méthode en PHP peut être définie en préfixant sa déclaration avec : public, protected, ou private.

Remarque:
Lorsque vous êtes à l'intérieur d'une méthode, PHP va automatiquement affecter l'objet à la variable $this pour pouvoir travailler avec les propriétés de la classe correspondante ($this->title = $title;).

POO en PHP: Constructeur et destructeur

```
class Product {
 public $title;
 function __construct($title){//Appelée lors l'instanciation
 $this->title = $title;
 }

 function getTitle(){ … } //retourne le titre
 function __destruct(){…} //Appelée lors de la suppression de
l'instance

}
$p = new Product("Ford")
unset($p);
```

POO en PHP: Les méthode magiques

Les méthodes magiques sont des méthodes qui vont être appelées automatiquement dans le cas d'un évènement particulier.

Les méthodes magiques reconnaissables en PHP au fait que leur nom commence par un double underscore __:

Fonction	Description
__construct()	Constructeur, appelé lors de l'instanciation d'un objet.
__destruct()	Destructeur, appelé lorsque l'objet est détruit ou en fin de script.
__call()	Intercepte les appels à des méthodes **inaccessibles** (privées/inexistantes).
__callStatic()	Intercepte les appels à des **méthodes statiques inaccessibles**.
__get()	Intercepte l'accès à une **propriété inexistante** ou **privée**.
__set()	Intercepte l'assignation à une **propriété inexistante** ou **privée**.
__isset()	Déclenché par isset() ou empty() sur une **propriété inexistante**.
__unset()	Déclenché par unset() sur une **propriété inexistante**.
__toString()	Définit la conversion de l'objet en **chaîne de caractères** (echo $objet).
__invoke()	Permet d'appeler un objet comme une **fonction**.
__clone()	Déclenché lors de la **clonage (clone) d'un objet**.
__sleep()	Appelé avant la **sérialisation (serialize())**, spécifie les propriétés à enregistrer.
__wakeup()	Appelé lors de la **désérialisation (unserialize())**.
__serialize()	Personnalise la **sérialisation** en PHP 7.4+.
__unserialize()	Personnalise la **désérialisation** en PHP 7.4+.
__debugInfo()	Définit les informations retournées par var_dump().

POO en PHP: Héritage

- Une classe peut hériter des méthodes et des membres d'une autre classe en utilisant le mot-clé extends dans la déclaration. L'héritage multiple n'est pas supporté en PHP.

- Les méthodes et membres hérités peuvent être surchargés en les redéclarant avec le même nom que dans la classe parente. Si la classe parente a défini une méthode comme **final**, alors celle-ci ne sera pas surchargeable.

Exemple:

```
class Electrique extends Voiture {
 public $volt;
 function __construct($marque, $puissance, $volt){
 parent::__construct($marque, $puissance);
 $this->volt = $volt;
 }
```

Exercice d'application:

1. Créez une classe **Vehicule** avec les propriétés suivantes :

Marque du véhicule, modèle du véhicule, année de fabrication, couleur du véhicule, type de carburant.

2. Ajoutez un constructeur à la classe Vehicule qui accepte les valeurs pour toutes les propriétés et les initialise.

3. Ajoutez des méthodes à la classe Vehicle :

start() : Une méthode qui affiche un message disant que le véhicule démarre.
stop() : Une méthode qui affiche un message disant que le véhicule s'arrête.

Tester le fonctionnement de l'objet de les méthodes crées.

Gestion des erreurs:

Les erreurs en PHP sont gérer par le système **error reporting** qui affiche sur la page la gravité des messages : Warning, Notice, Fatal error.

PHP attribue pour chaque niveau d'erreur un numéro et une constante qui représente un type d'erreur.

Exemples:

1 : E_ERROR : erreur critique entraînant une interruption du script.

2 : E_WARNING : message d'avertissement qui peut entraîner des comportements anormal par la suite (comme l'échec de l'ouverture d'un fichier) .. mais PHP peut continuer son exécution.

Etc.

Gestion des erreurs: Gestionnaire d'erreurs personnalisé

Le principe consiste à appeler une fonction lorsqu'une erreur PHP se produit. Cette fonction peut accepter cinq paramètres :

- **$errno** – le niveau d'erreur (obligatoire)
- **$errstr** – le message (obligatoire)
- **$errfile** – le nom du fichier (optionnel)
- **$errline** – le numéro de ligne (optionnel)
- **$errcontext** – un tableau avec toutes les variables qui existaient lorsque l'erreur a été déclenchée (optionnel).

```
function ErrorPersonnalise($errno, $errstr, $errfile, $errline) {
        echo "Erreur numéro [$errno], ligne [$errline], du fichier $errfile :  $errstr";
    }
// appel de la fonction 'ErrorPersonnalise' si erreur
set_error_handler(ErrorPersonnalise);
```

Gestion des erreurs: Les exceptions (identiques à C++/Java)

Depuis la version 5, PHP propose une nouvelle gestion des erreurs plus simple et plus personnalisable. Il suffit d'imbriquer l'instruction pouvant provoquer une erreur dans un bloc **try** et de récupérer l'erreur dans le bloc **catch**.

```
try {
        // Code susceptible de générer une erreur
    } catch (Exception $e) {
        // Gérer l'erreur
        echo 'Une erreur s\'est produite : ' . $e->getMessage();
    } finally{
        // Code qui est toujours exécuté
    }
```

Remarque: Pour générer une erreur on utilise l'instruction:

```
throw new Exception('Message d\'erreur à transmettre');
```

Gestion des erreurs: Terminer l'exécution du script

Dans certains cas, il n'est ni possible ni utile de poursuivre l'exécution du code PHP (variables non définies, valeurs erronées, échec de connexion, …)

Arrêt brutal de l'exécution du code:

```
die(message);
exit(message);
```

Envoie message au navigateur et termine l'exécution du script courant.

Exercice d'application:

1. Lancer XAMP Control Panel puis démarrer Apache et MySQL.

2. Ecrire un script php permettant de faire la division de deux nombres passés comme des arguments au script.

3. Traiter le cas de division par 0 en utilisant try catch. Pourquoi le script s'arrête ?

4. Générer une exception pour que le script ne soit pas s'arrêter

Accès aux base de données:

Le langage PHP support la majorité des SGBD.

Parmi les plus connues, on peut citer :

MySQL, **SQLite**, PostgreSQL, Oracle, Ingres, Interbase, Informix, Microsoft SQL Server, mSQL, Sybase, FrontBase, dBase, etc ...

Accès aux base de données:

1. Connexion au SGBD.
2. Sélection d'une base.
3. Envoi d'une requête.
4. Récupération et utilisation du résultat.
5. Fermeture de la connexion.

Remarque:

On peut itérer les étapes 3 et 4 autant de fois que l'on veut avant de fermer la connexion à l'étape 5.

Accès aux base de données: MySQL

PHP offre 2 APIs différentes pour se connecter à MySQL :

- mysqli : cette extension permet d'accéder aux fonctionnalités fournies par MySQL 4.1 et supérieur. En plus d'une interface orientée objet, mysqli propose aussi une interface par fonctions.

- PDO : PDO (PHP Data Objects) fournit une interface d'abstraction à l'accès de données, ce qui signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données quelque soit la base de données utilisée.

Accès aux base de données: MySQL

PHP fournit un grand choix de fonctions permettant de manipuler une base de données MySQL. Toutefois, parmi celles-ci quatre fonctions sont essentielles :

- La fonction de connexion au serveur (mysqli_connect ou mysqli_real_connect)
- La fonction de choix de la base de données (mysqli_select_db)
- La fonction de requête (mysqli_query)
- La fonction de déconnexion (mysqli_close)

Accès aux base de données: MySQL

L'exécution d'une requête SELECT avec mysqli_query() retournera un objet résultat de type mysqli_result (ou TRUE pour les autres types de requêtes).

Les fonctions de traitements de résultat d'une requête sont au choix:

- mysqli_fetch_row()
- mysqli_fetch_array()
- mysqli_fetch_assoc()
- mysqli_fetch_object()
- mysqli_free_result()

Accès aux base de données: MySQL

Exemple: Connexion mysqli

```php
define('SERVER', 'localhost');
define('USER', 'root');
define('PASSWORD', 'test');
define('DB', 'test');

$conn = mysqli_connect(SERVER, USER, PASSWORD, DB);
if (!$conn) die('Echec de connexion au serveur');

echo 'Connection avec succès ... ' . mysqli_get_host_info($conn);

//ou mysqli comme objet
$conn = new mysqli(SERVER, , USER, PASSWORD, DB);
```

Accès aux base de données: MySQL

Exemple: Connexion PDO

```
define('SERVER', 'localhost');
define('USER', 'root');
define('PASSWORD', 'test');
define('DB', 'test');

$pdo_db = new PDO('mysql:host=' . SERVER . ';dbname=' . DB, USER, PASSWORD )
       or die("Echec de la création de l'instance PDO !");

echo "Classe PDO : succès ... <br />";
```

Accès aux base de données: MySQL

Exemple: Insertion et modification

```
//mysqli
$result = mysqli_query($conn, "INSERT INTO user VALUES('samir','kamal',12)");
if ($result) echo 'Ajoute avec succès\n';
else echo 'Problème de connexion\n’;


//PDO
$result = $pdo_db->query("INSERT INTO user VALUES('amal','amal',14)");
if ($result) echo 'Ajoute avec succès';
else echo 'Problème de connexion';
```

Accès aux base de données: MySQL

Exemple: récupération

```
//mysqli
$result = mysqli_query($conn, "SELECT * FROM user ");
    if ($result) {
        while ($row = mysqli_fetch_array($result)) {
            echo "{$row["nom"]} - {$row["prenom"]}<br />";
        }
    } else echo 'Problème de connexion';
```

Accès aux base de données: MySQL

Exemple: récupération

```php
//PDO
$result = $pdo_db->query("SELECT * FROM user ");
    if ($result) {
        while ($row = $result->fetch()) {
            echo "{$row["nom"]} - {$row["prenom"]}<br />";
        }
    } else echo 'Problème de connexion';
```

Accès aux base de données: MySQL

Exemple: fermeture de connexion

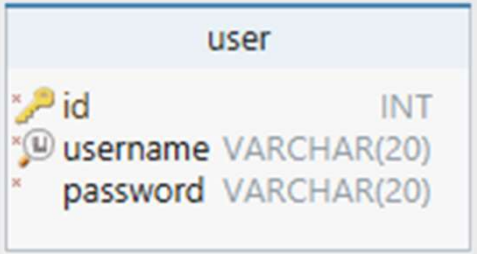
```php
// Fermeture de la connexion (n'est pas nécessaire mais peut être fait si besoin)

//mysqli
mysqli_close($conn);


//PDO
$pdo = null;
```

Exercice d'application:

1. A l'aide de phpMyAdmin, créer une base de données nommées testbd
2. Ajouter à cette base de données la table suivante:
3. Se connecter à la base de données avec mysqli_connect
4. Insérer les données suivantes:
   - (1, test, 1234)
   - (2, test1, 4321)
5. Afficher les données insérées



Les formulaires:

Intérêt d'un formulaire

- Les formulaires sont les éléments essentiels qui permettent de créer une interactivité entre un site et ses visiteurs.
- Tout échange entre visiteur et serveur passe par un formulaire, dans lequel l'utilisateur peut saisir textes, un mots de passe, …

Création d'un formulaire:

```
<form action='reception_formulaire.php' method='GET ou POST'>
    <!-- différents champs -->
</form>
```

Les formulaires: Traitement

Les attributs de l'élément form

- action : désigne la page vers laquelle seront envoyées les informations rentrées.
- method: méthode d'envoi des données vers le serveur.

Les méthodes d'envoi:

- Deux méthodes existent GET et POST
- GET: utilisée par défaut, données transmises visibles par l'utilisateur

  (http://..../traitement.php?prenom=ahmed&nom=kamal)
- POST: données non visibles mais pas de navigation avec Précédent/Suivant.
- PHP associe deux tableaux $_GET et $_POST pour récupérer les données passées.

Les formulaires: Traitement (méthode POST)

Exemple:

```
<form action="valider.php" method="post">
    <input name="genre" type="text" />
    ...
</form>
```

- Ce code HTML spécifie que les données du formulaire seront soumises à la page web “valider.php” en utilisant la méthode POST.
- Prenez soin de noter les noms (après name) des données du formulaire, car ils représentent les "clés" dans le tableau associatif "$_POST".
- $_POST[‘genre’] permettra de récupérer la valeur du champ genre.

Les formulaires: Traitement (méthode GET)

Exemple:

```
<form action="valider.php" method="get">
    <input name="genre" type="text" />
    ...
</form>
```

- La différence avec la méthode POST est qu'elle passe les variables à la page web "valider.php" en les ajoutant à la fin de l'URL.
- Après avoir cliqué soumettre, l'URL aura ceci ajouté à la fin : "valider.php?genre=xxx"
- Le point d'interrogation "?" dit au navigateur que les objets suivants sont des variables.
- $_GET['genre'] permettra de récupérer la valeur du champ genre.

Exercice d'application:

1. Créer une page html contenant un formulaire avec trois inputs (text, password et submit).

2. Ecrire un script **save.php** qui permet de récupérer les informations de la page précédente et les insérer dans la table **user** de la base de données **testdb**.

Les cookies:

Un cookie est un fichier que le serveur envoi sur la machine de l'utilisateur. Il est souvent utilisé pour reconnaitre les utilisateurs.

Les cookies sont utiles, par exemple, pour mémoriser les identifiants de connexion d'un internaute pour éviter de les ressaisir à chaque fois .

L'utilisation des cookies permet aussi à PHP de faire fonctionner les sessions.

Les cookies: Création

La fonction setcookie(name, value, expire, path, domain) est utilisé pour créer 1 cookie.

```
<?php
setcookie("visite", 1, time()+365*24*3600); // expiration au bout de
1 an
?>
```

Les cookies: Création

- La fonction setcookie() doit être placé avant tout code HTML, car le cache du navigateur doit être vide pour que cette fonction fonctionne convenablement.

- Pour connaitre la durée écoulée jusqu'à aujourd'hui, on utilise la fonction time().

Les cookies: Modification

- Afficher un cookie: `$_COOKIE["visite"]`
- Modifier la valeur d'un cookie: `setcookie("visite",2, time()+3600);`
- Supprimer un cookie: `setcookie("visite");`

Les sessions:

Le transfert d'information d'une page (script) à un autre n'est pas simple en PHP.

Le protocole HTTP ne permet pas de conserver les connections.

Les solutions:

1) Les formulaires : pas simple, pas sécurisé, pas toujours possible
2) Les cookies : pas simple, non sécurisé
3) Les sessions : mécanisme dédié interne à PHP

Les sessions:

- Les sessions permettent de stocker des informations côté serveur.
- Elles sont identifiées par un numéro qui reste valide tant que le visiteur reste connecté.
- Les données se placent et se récupèrent dans $_SESSION.

Les sessions: Utilisation

La session existe dès qu'elle est crée et jusqu'à ce qu'elle soit détruite.

- Création (et réouverture) : `session_start()`
- Destruction : `session_destroy(), session_unset()`
- Modification: `$_SESSION`

Remarque : les sessions s'autodétruisent après un certain temps (généralement 30 min).

Exercice d'application:

1. Ecrire un script **session_test.php**:
   - **Démarrer** la session
   - Se connecter à la base de données **testdb**
   - Récupérer le nom d'utilisateur de premier utilisateur et l'ajouter à la variable superglobale **$_SESSION**.

2. Ecrire un script **session_show.php** :
   - **Démarrer** la session
   - Afficher le nom d'utilisateur existant dans le tableau **$_SESSION**

3. Accéder au fichier **session_test.php** avec le navigateur puis aller au fichier **session_show.php**. Qu'est-ce que vous remarquez ?

Les fichiers:

PHP fournit de nombreuses fonctions pour manipuler des fichiers:
Les appels de base pour la gestion des E/S fichiers sont : fopen, fread, fwrite, fclose, ...

Ovrire un fichier: `$file = fopen(chemin, mode);`

Le mode peut prendre comme Valeur:

- 'r' : lecture (read)
- 'w' : écriture (write)
- 'a' : ajout (append)
- '+' : lecture/´ecriture

Exemple:

```
if (!($f = fopen("exemple.txt", "r")))
exit("impossible d'ouvrir le fichier!");
```

Les fichiers:

- Lecture caractère par caractère: `$car=fgetc($file);`
- Lecture de lignes : `$ligne=fgets($file, $nbr_lignes);`
- Lecture d'octets: `$v=fread($file, $nbr_octets);`
- Taille d'un fichier: `$taille = filesize('chemin');`
- Ecriture: `fwrite($file, $message, [$longueur_maximale]);`
- Fin de fichier: `feof($file);`
- Fermeture: `fclose($file);`
- Existence d'un fichier :`file_exists('chemin');`
- `file_get_contents` et `file_put_contents` permettent de lire et d'écrire le contenu d'un fichier de façon rapide.

Les fichiers: Require & Include

Il est ensuite possible d'inclure et d'exécuter un fichier à l'aide des instructions:

```
include("fichier.php");
require("fichier.php");
include_once("fi.php");
require_once("fi.php");
```

- **require** erreur bloquante
- **include** erreur non bloquante
- **…_once** pour les inclusions uniques

Exercice d'application:

1. Ecrire un script **connect.php** qui contient une fonction permettant de se connecter à la base de données **testdb**.

2. Ecrire un script **insert.php** qui utilise la fonction du fichier **connect.php** pour insérer dans la table **user** la donnée suivante (5, test5, 5555).

3. Ecrire un script **save.php** qui **sauvegarder** la donnée de la table user dans un fichier texte **users.txt.**