PHP – Développement Web

CM3 : PHP « avancé »

Mickaël Martin Nevot

V1.16.0

PHP – Développement Web

Structure de langage

- **Structures de langage** :
  - `echo`, `return`, `include`, `require`, etc.

≠ fonction

- Utilisation de parenthèses non conseillée (même avec plusieurs arguments) :
  - Gain de performances :

```php
echo $a; // Plutôt que echo ($a);
echo '1, ', '2'; // Plutôt que echo '1, ' . '2';
```

  - Erreur en cas de plusieurs arguments :

```php
// Cause une erreur.
echo ($a, $b);
```

  - Conversion en expression :

```php
// Erreur vous voulez retourner une référence !
return ($a);
```

Retour sur return !

- Depuis une fonction : termine et retourne l'argument
- Depuis un script inclus : retourne par la fonction d'inclusion au script appelant
- Depuis le script principal : arrêt du script

Quelques fonctions

- Teste la **définition d'une variable** :
  - `isset($var_1, $var_2, …)`
- Savoir si une variable est vide : `empty($var)`
- Crée un **MD5** : `md5($str, …)`
- Génère un **identifiant unique** : `uniqid($prefix = …, …)`
- Génère un **nombre aléatoire** : `mt_rand(…)`
- Envoie un **en-tête HTTP** : `header($str, …)`
  - Redirection du navigateur :
    - `$str = 'Location: ' . $url`
- Initialise une **session** : `session_start()`

Pas d'affichage avant

Important !

Avant d'utiliser un header

GET et POST

- HTTP (*hypertext transfer protocol*) :
  - Protocole de communication (client-serveur)
  - Méthodes (commandes) :
    - **GET** : construction d'URL dynamique
    - **POST** : passage « caché » des valeurs

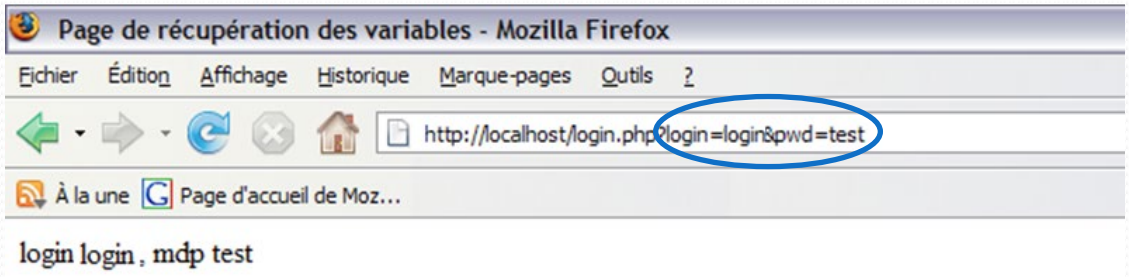
URL (*uniform resource locator*) : littéralement « localisateur uniforme de ressource », est une chaîne de caractères utilisée pour adresser les ressources du Web, aussi appelée **adresse Web**

- Variables « superglobales » : `$_GET`, `$_POST`
  - Récupération de valeurs passées via le protocole HTTP et les méthodes **GET** et **POST**

GET et POST

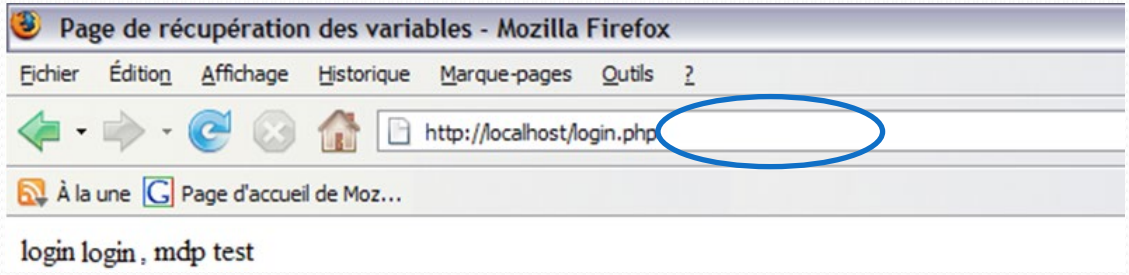
- **GET :**

```
if (isset($_GET['login'], $_GET['pwd']))
{
    echo 'login ', $_GET['login'], ', mdp ', $_GET['pwd'];
}
```



- **POST :**

```
if (isset($_POST['login'], $_POST['pwd']))
{
    echo 'login ', $_POST['login'], ', mdp ', $_POST['pwd'];
}
```



E-mail

- Fonction :
  - `mail($to, $subject, $message, $headers = ..., ...)`
- Envoi d'un *e-mail* :
  - Destinataire(s) : `$to`
  - Sujet du mail : `$subject`
  - Message à envoyer : `$message`
    - Texte simple
    - HTML
    - Texte simple et HTML (multi-format)
  - En-têtes : `$headers`

E-mail multi-format

- *E-mail* en texte **simple** :
  - Toujours lisible : compatibilité totale
  - Très léger
- *E-mail* en **HTML** :
  - Mise en forme complexe (police, image, lien, etc.)
- *E-mail* **multi-format** (**meilleure solution**) :
  - Affichage en texte simple en cas d'incompatibilité HTML
  - Affichage en HTML autrement

En-tête et frontière

- En-tête :
  - Adresse de l'expéditeur : `'From: exp1@fai.fr'`
  - Adresse de réponse : `'Reply-To: exp2@fai.fr'`
  - Destinataire(s) en copie : `'Cc: email_1, email_2'`
  - Destinataire(s) en copie cachée : `'Bcc: email_1, email_2'`
  - *Internet media type* : `'Content-type: type; sous-type'`
- Frontière (*boundary*) :
  - Permet de faire la séparation texte simple / HTML
  - *Content-type* de l'entête :
    `'Content-type: multipart/alternative; boundary=frontière'`
  - Au début de chaque type de message :
    - Frontière préfixée de deux tirets accolés
    - Après la frontière, modification des *headers* appliqués localement (au moins le *content-type* du message)

Peut être fausse

Exemple d'e-mail

```php
$to = 'to_mail@fai.org';
$from = 'my_mail@fai.org';
$reply = 'reply_mail@fai.org';
$subject = 'Test mail';

$bndary = md5(uniqid(mt_rand()));
$headers = 'From: Name <' . $from . '>' . "\n";
$headers .= 'Return-Path: <' . $reply . '>' . "\n";
$headers .= 'Content-type: multipart/alternative; boundary="' . $bndary. '"';

$message_text = 'Hello world!';
$message_html = '<html><body><strong>Hello world!</strong></body></html>';
$message = '--' . $bndary . "\n";
$message .= 'Content-Type: text/plain; charset=utf-8' . "\n\n";
$message .= $message_text . "\n\n";
$message .= '--' . $bndary . "\n";
$message .= 'Content-Type: text/html; charset=utf-8' . "\n\n";
$message .= $message_html . "\n\n";

mail($to, $subject, $message, $headers);
```

Fichier

- **Ouverture** : `fopen($file, $mode, ...)`
  - Fichier à ouvrir : `$file`
  - Spécifie le type d'accès désiré au fichier : `$mode`
    - Ouvre un fichier à son début : `'r'`, `'r+'`
    - Ouvre un fichier à sa fin : `'a'`, `'a+'`
    - Crée/écrase et ouvre un fichier : `'w'`, `'w+'`
    - Crée et ouvre un fichier : `'x'`, `'x+'`
- **Fermeture** : `fclose($handle)`

renvoie un pointeur sur le fichier

Ouvre en lecture dans tous les cas et en plus en écriture en cas de présence de + dans le mode

Fichier

- **Lecture** (jusqu'à `$length` octets du fichier `$handle`, ou jusqu'à la fin de la ligne courante par défaut pour `fgets(...)`) :
  - `fread($handle, $length)`
  - `fgets($handle, $length = ...)`
- **Écriture** :
  - `fwrite($handle, $str, ...)`
  - `fputs($handle, $str, ...)`
- Test si fichier/répertoire existe : `file_exists($filename)`
- Copie du fichier : `copy($source, $dest, ...)`
- Renomme un fichier/répertoire : `rename($old, $new, ...)`
- Efface un fichier : `unlink($filename)`

Nombre de caractères à lire

Valeur retournée par `fopen()`

`fputs(...)` est un alias de `fwrite(...)`

Chaine de caractères à écrire

Chargement HTML d'un fichier

- Le nom original du fichier (sur la machine du client) :
  `$_FILES['variable']['name']`
- Le nom temporaire du fichier (sur le serveur) :
  `$_FILES['variable']['tmp_name']`
- *L'Internet media type* du fichier :
  `$_FILES['variable']['type']`
- La taille du fichier (en octets) :
  `$_FILES['variable']['size']`
- L'éventuel code d'erreur :
  `$_FILES['variable']['error']`

```
<form action="send_fichier.php" method="post" enctype="multipart/form-data">
    <p>
        Votre fichier : <input type="file" name="mon_fichier"></input>
        <input type="hidden" name="MAX_FILE_SIZE" value="20000"></input>
    </p>
</form>
```

Variables dynamiques

- Nom de variable affecté et utilisé dynamiquement :

```php
$var = 'hello';
$hello = 'world';
echo ${$var}; // Affiche : world.

…

$champ_cache = $_POST['champ_cache'];

function insert() { // Code. }
function delete() { // Code. }
function update() { // Code. }
function select() { // Code. }

if ('insert' == $champ_cache
 || 'delete' == $champ_cache
 || 'update' == $champ_cache
 || 'select' == $champ_cache)
{
    $champ_cache();
}
```

PHP et MySQL

BD : base de données

- **Connexion** à un serveur MySQL :
  `mysqli_connect($host = …, $username = …, $passwd = …, …)`
- Sélectionne une **BD** :
  `mysqli_select_db($link, $dbname)`
- Ferme la connexion MySQL :
  `mysqli_close($link)`
- Envoie un message en cas d'erreur : `die($str)`

Une connexion est fermée par défaut à la fin du script l'ayant ouverte

```
$link = mysqli_connect($host, $username, $passwd) or die('Erreur : ' . $host);
```

Important : l'activation de l'extension MySQLi est obligatoire (par défaut à partir de PHP 5.3.0) !

PHP et MySQL

- Envoie une **requête** au serveur :
  `mysqli_query($link, $query)`
- Retourne un **enregistrement** :
  - Tableau indexé :
    `mysqli_fetch_row($result)`
  - Tableau associatif :
    `mysqli_fetch_assoc($result)`
- Nombre d'enregistrements d'un résultat :
  `mysqli_num_rows($result)`
- Libère la mémoire du résultat :
  `mysqli_free_result($result)`

Exemple PHP et MySQL

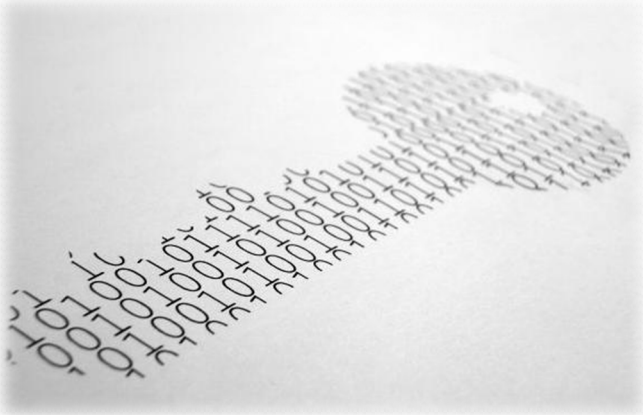
```php
$link = mysqli_connect('localhost', 'mysql_username', 'mysql_passwd')
    or die('Pb de connexion au serveur: ' . mysqli_connect_error());
mysqli_select_db($link, 'my_dbname') or die ('Pb de sélection BD : ' . mysqli_error($link));

$query = 'SELECT name AS username FROM table WHERE id = 1';
$result = mysqli_query($link, $query);
if (!$result)
{
    echo 'Impossible d\'exécuter la requête ', $query, ' : ', mysqli_error($link);
}
else
{
    if (mysqli_num_rows($result) != 0)
    {
        while ($row = mysqli_fetch_assoc($result))
        {
            echo $row['username'];
        }
    }
}
```

Utilité des pages sécurisées

- Restreindre une partie d'un site Web
- **Protéger des données**
- Utilisations courantes :
  - Partie d'un site accessible qu'à certains utilisateurs
  - Partie d'un site qui comporte des **données personnelles** (compte bancaire, adresse postale, etc.)
  - Compte ***e-mail***
  - Compte pour un site *d'**e-commerce***

Moyens de sécurisation

- Niveau **réseau** : (par exemple HTTPS)
  - On utilise SSL (*secure socket layer*) pour chiffrer le canal de communication :
    - Utilise un certificat pour l'identification
    - Certificat hébergé chez un tiers
- Niveau **applicatif** :
  - Authentification
  - Pages sécurisées



Authentification

- Table **utilisateur** dans la base de données :
  - Un identifiant (*e-mail*, pseudonyme, etc.)
  - Un mot de passe encodé
- Lors de l'**authentification** :
  - On vérifie le couple identifiant / mot de passe encodé avec les informations en base de données
  - On met les informations dans la variable `$_SESSION`
- **Vérification de l'authentification à chaque page** :
  - L'utilisateur ne retape pas ses identifiants
  - Vérification faite grâce à la variable `$_SESSION`

Exemple d'authentification

```php
// Page d'authentification.
if (filter_input(INPUT_POST, 'login') && filter_input(INPUT_POST, 'pwd'))
{
    // Connexion au serveur de la base de données, à la base de données et
    // récupération dans la table utilisateur du login dans la variable $login
    // et du mot de passe dans la variable $pwd.
    // …
    if ($login == $_POST['login'] && password_verify($_POST['pwd'], $pwd))
    {
        session_start();
        $_SESSION['suid'] = session_id();
        header('location: page.php');
    }
    else
    {
        header('location: index.html');
    }
}
```

Exemple d'authentification

```php
// page.php.
session_start();
if (isset($_SESSION['suid']))
{
    // L'authentification est validée.
}
```

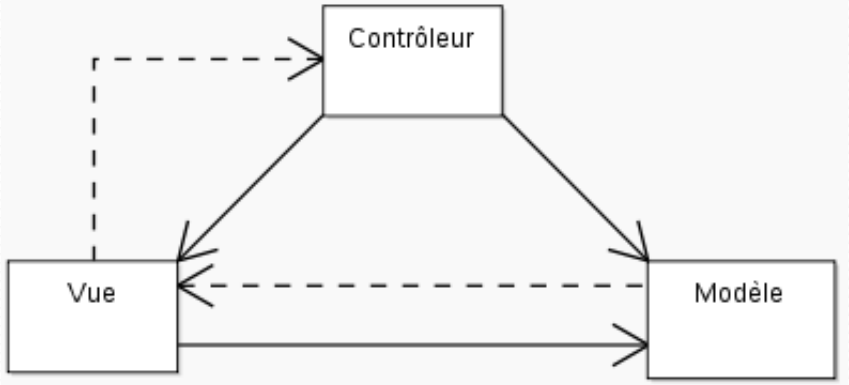
Frameworks

- Espace de travail modulaire ← Également appelé cadriciel
- **Ensemble de bibliothèques, d'outils et de conventions permettant le développement d'applications**
- Permet de produire une application aboutie et **facile à maintenir**
- Composants organisés pour être utilisés en interaction les uns avec les autres
- Guide architectural
- Impose une rigueur de développement

Frameworks

- Concepts fondamentaux :
  - **ORM** (*object-relational mapping*) : *mapping* objet-relationnel

    Correspondance entre monde objet et monde relationnel
  - Modèle **MVC** : Modèle-vue-contrôleur
    - **Architecture** et **méthode de conception** d'IHM
    - **Modèle** : données et leur manipulation
    - **Vue** : élément de l'interface graphique
    - **Contrôleur** : orchestre les actions, synchronise
    - **MVC 2** : un seul contrôleur



Frameworks

- Principaux *frameworks* PHP :
  - **Zend framework**
  - **Symfony**
  - CakePHP
- **Avantages** :
  - Maintenance facilitée
  - Architecture logicielle propre
- **Inconvénients** :
  - Nécessite de fortes compétences (emploi de spécialistes / ingénieurs)
  - Performances diminuées

Aller plus loin

- Pièces jointes dans un *e-mail*
- Flux d'entrée/sortie
- Utiliser la librairie GD (traitement des images)
- PDO
- Web 3.0 ?
  - Web sémantique
  - Web 3D
  - Web hors navigateur (applications de bureau, *smartphone*)
- Etc.

Liens

- Documents électroniques :
  - Manuels :
    - http://php.net/manual
    - http://fr.php.net/manual/fr/ref.mysql.php
  - *Framework* :
    - http://framework.zend.com
    - http://www.symfony-project.org
  - Génération de données :
    - http://www.generatedata.com/#generator
  - Fonctions images :
    - http://www.manuelphp.com/php/ref.image.php

Crédits

Auteur

Mickaël Martin Nevot
mmartin.nevot@gmail.com

Carte de visite électronique

Relecteurs

Cours en ligne sur : www.mickael-martin-nevot.com