Cours Web - PHP

Thierry Vaira

BTS La Salle Avignon

Sommaire


APACHE
HTTP SERVER


php


W3C

Objectifs

- Se familiariser avec le langage PHP et acquérir une pratique minimale.
- Être capable de réaliser des scripts serveurs pour un site web en respectant les bonnes pratiques.
- Acquérir les capacités d'auto-formation nécessaires pour suivre les évolutions à venir.

Le langage PHP (*Personnal Home Page*) a été créé par Rasmus Lerdorf en 1994 pour des besoins personnels. En 1997, le projet devient un travail d'équipe et l'interpréteur est réécrit par Zeev Suraski et Andi Gutmans pour donner la version PHP3, version qui s'est rapidement imposée et devient PHP (*PHP Hypertext Preprocessor*).

PHP : Langage de programmation de scripts

- PHP est un **langage de programmation de scripts côté serveur** permettant de **produire des pages web dynamiques**.

L'utilisation de PHP en tant que générateur de pages Web dynamiques est la plus répandue, mais il peut être utilisé aussi comme langage de programmation ou de script en ligne de commande (**CLI**) sans utiliser de serveur HTTP ni de navigateur.

PHP : Langage impératif et objet

- C'est un **langage impératif** disposant depuis la version 5 de fonctionnalités de **modèle objet** complètes.

Un **langage impératif** est un langage de programmation qui met l'accent sur les modifications des variables provoquées par l'exécution d'instructions.
La **programmation orientée objet** consiste à définir des objets logiciels et à les faire interagir entre eux.

PHP : Langage à typage dynamique faible et souple

- C'est un langage à **typage dynamique faible et souple**, donc facile à apprendre par un débutant mais, de ce fait, des failles de sécurité peuvent rapidement apparaître dans les applications.

Le **typage** d'une variable consiste à associer à son nom un « type » de donnée, permettant à l'ordinateur de savoir si celle-ci est de type numérique (int, float, ...), textuel (char, string, ...), etc ... Généralement, on « type » la variable au moment de sa déclaration.
Le **typage dynamique** consiste à laisser l'ordinateur réaliser cette opération de typage « à la volée » lors de l'exécution du code. C'est donc l'affectation d'une **valeur** qui donnera le type à la variable.

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 appartient à la grande famille des descendants du C, dont la syntaxe est très proche. En particulier, sa syntaxe et sa construction ressemblent à celles des langages Java et Perl.

Utilisation

- Généralement, PHP sert :
  - à **produire des pages web dynamiques** et donc
  - à **récupérer et traiter des informations issues d'une base de données, d'un système de fichiers** (contenu de fichiers et de l'arborescence) ou plus simplement **des données envoyées par le navigateur**.

En 2002, PHP est utilisé par plus de 8 millions de sites Web à travers le monde et en 2007 par plus de 20 millions. Plus d'un quart des vulnérabilités répertoriées concerne des applications PHP. PHP est utilisé sur des sites Web à très fort trafic comme Yahoo, Facebook, Wikipédia, ...
Aujourd'hui, près de 80% des sites internet utilisent le langage PHP sous ses différentes versions. Plusieurs développeurs PHP responsables de ces sites utilisent en majorité la version 5.4 (38%) dans leurs missions quotidiennes.

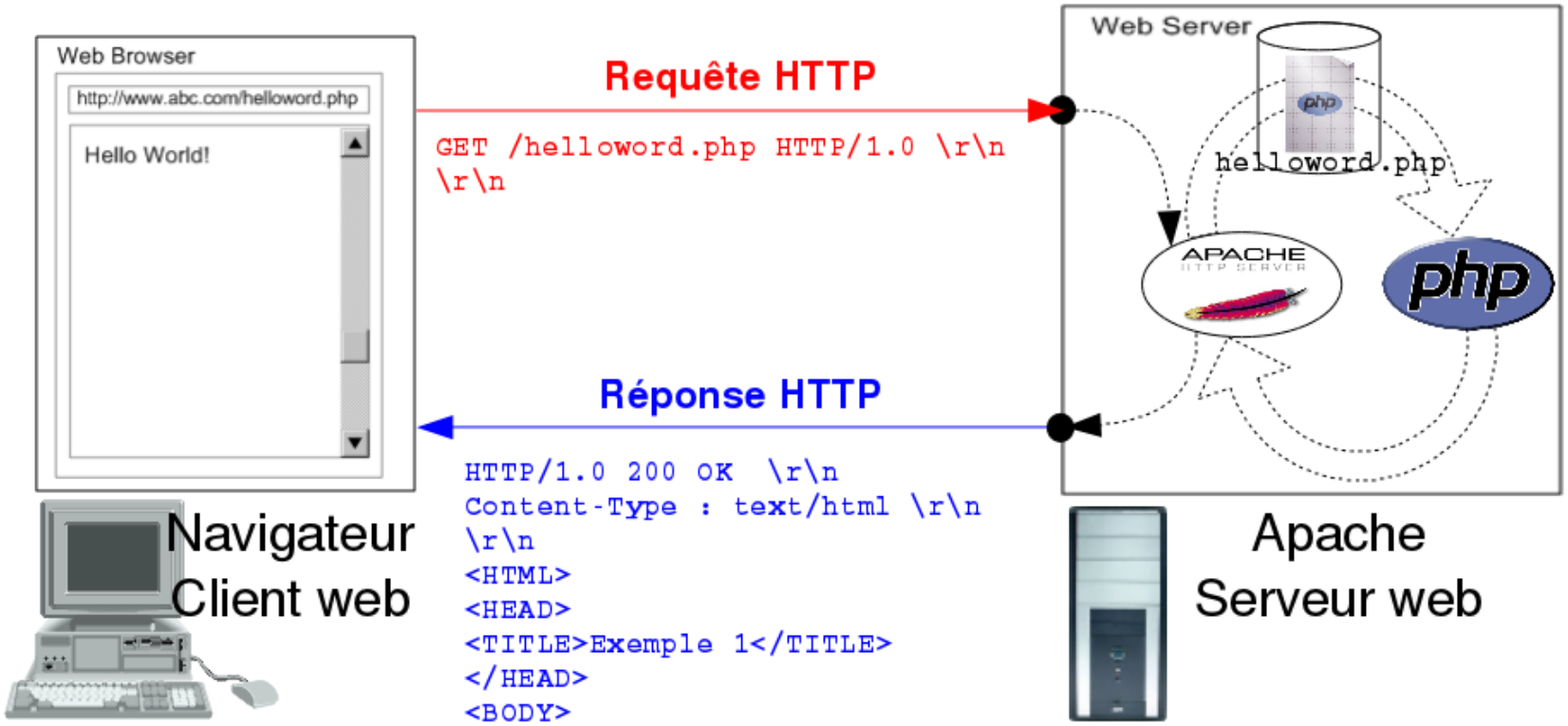
Exemple : le traditionnel *Hello world*

➠ Le **code PHP** doit être inséré entre des **balises** `<?php` et `?>`. Le **script** doit porter l'extension `.php`.

Le script `helloworld.php` :

```
<HTML>
   <HEAD>
      <!-- Un titre -->
      <TITLE>Exemple 1</TITLE>
   </HEAD>
   <BODY>
    <?php
    // forme la plus simple, recommandée
    echo '<P>Hello World!</P>';
    ?>
   </BODY>
</HTML>
```

Fonctionnement



Client/Serveur I

➠ Pour réaliser un développement PHP, il vous faut la chaîne complète client/serveur HTTP. Plusieurs solutions s'offrent à vous :

- le serveur est présent en local sur votre machine de développement (*localhost*). Le plus souvent sous Linux, la racine des documents du serveur se trouvent en `/var/www/` . Et l'accès par le navigateur se fait à l'adresse : http ://localhost/ ou http ://votre-adresse-ip/
- le serveur est présent sur l'intranet de votre structure de développement (entreprise, école, université, domicile, ...). Le serveur de la section est configuré pour un accès pour chaque compte. La racine se trouve dans votre répertoire personnel `$HOME/public_html/` et l'accès client se fait par exemple à l'adresse : http ://192.168.52.85/~$LOGIN/

Client/Serveur II

- le serveur est présent sur l'Internet, le plus souvent chez un hébergeur. Dans ce cas, il faut transférer les documents de votre poste de développement vers le serveur Internet (le plus souvent en FTP).

LAMP est un acronyme désignant un ensemble de logiciels libres permettant de construire des serveurs de sites web. L'acronyme original se réfère aux logiciels suivants : **L**inux (l'OS GNU/Linux), **A**pache (le serveur Web), **M**ySQL (le serveur de base de données) et **P**HP (le langage de script). Il existe aussi une architecture **WAMP** utilisée pour développer des sites web sur une machine *Windows*.

Sommaire

Syntaxe de base

➠ Le code PHP est composé par des appels à des **fonctions**, dans le but d'affecter des **valeurs** à des **variables**, le tout encadré dans des **conditions**, des **boucles**.

- Les instructions sont séparées par des "**;**"
- les variables sont toujours préfixées par un **$**
- Les blocs d'instructions sont délimités par les caractères "**{**" et "**}**"

PHP supporte les commentaires de type C, C++ et Shell Unix (aussi appelé style Perl) : **//**, **/* */** et **#**.

Manuel PHP : https ://php.net/manual/fr/

Exemple : un contenu dynamique

```php
// URL : http://localhost/helloworld-2.php?lang=fr
$lang = strtolower($_GET['lang']); // appel de la fonction strtolower()

if ($lang === 'fr') // test du type ET de la valeur
{
    $message = 'Bonjour le monde !'; // une chaîne de caractères protégée
}
elseif ($lang == 'en') // test de la valeur seulement
{
    $message = "Hello"." World!"; // une concaténation de chaînes de caractères
}
else
{
    $message = "Je ne vois pas quelle est votre langue $lang !"; // une chaîne de
          caractères non protégée
}

echo $message; // affiche le contenu de la variable message
```

Exemple : le script indispensable

➠ En appelant la fonction `phpinfo()`, on obtient beaucoup d'informations intéressantes sur le système et sa configuration comme les variables pré-définies disponibles, les modules PHP chargés ainsi que la configuration.

```php
<?php

phpinfo();

?>
```

Les types

➠ PHP supporte **8 types basiques** (`boolean`, `integer`, `float` ou *double* et `string`), **2 types composés** (`array` et `object`) et **2 types spéciaux** (`ressource` et `NULL`).

- Pour afficher le type et la valeur d'une expression, utilisez la fonction `var_dump()` (ou `print_r()`)
- Pour afficher seulement le type à des fins de déboguage, utilisez la fonction `gettype()`
- Pour vérifier un certain type, utilisez les fonctions *is_type* (`is_int()`, `is_string()`, ...)

Les tableaux

- Un tableau en PHP est en fait une **carte ordonnée**.
- Une **carte** est un type qui associe des **valeurs à des clés**.

Le type `array` est optimisé pour différentes utilisations :

- il peut être considéré comme un tableau, une liste, une table de hashage, un dictionnaire, une collection, une pile, une file d'attente et probablement plus
- on peut avoir, comme valeur d'un tableau, d'autres tableaux, multidimensionnels ou non
- La clé *key* peut être soit un entier, soit une chaîne de caractères. Elle est optionnelle. La valeur *value* peut être de n'importe quel type.

☞ Il existe de nombreuses fonctions dédiés aux tableaux :
php.net/manual/fr/ref.array.php

Exemple : les tableaux

```php
// Par défaut la clé est un entier commençant à 0
$recette = array("250 g de farine", "4 oeufs", "1/2 l de lait", "1 pincée se sel
     " ,"2 cuillères à soupe de sucre", "50 g de beurre");
echo "<pre>"; print_r($recette); echo "</pre>";

for($i = 0; $i < count($recette); $i++)
{
    var_dump($recette[$i]);
    echo "<br />";
}

// Ici la clé est une chaîne de caractères
$liste = array("fruit" => "banane", "legume" => "radis", "boisson" => "eau");
echo "<pre>"; print_r($liste); echo "</pre>";

// La boucle foreach fournit une façon simple de parcourir des tableaux
foreach ($liste as $key => $value)
    echo "$key => $value"."<br />";
```

Les classes et objets

➠ Une **classe** peut contenir ses propres constantes, variables (appelées "propriétés" ou "**attributs**"), et fonctions (appelées "**méthodes**").

- Une **définition de classe** commence par le mot-clé `class`, suivi du nom de la classe puis d'une paire d'accolades contenant la définition des attributs et des méthodes appartenant à la classe.
- La pseudo-variable `$this` est disponible lorsqu'une méthode est appelée depuis un objet : `$this` est une référence à l'objet appelant. `$this` est obligatoire pour **accéder aux membres** de l'objet en utilisant l'opérateur `->`.
- Pour **créer une instance** d'une classe, le mot-clé `new` doit être utilisé. Un objet sera alors systématiquement créé, à moins qu'il ait un constructeur défini qui lance une exception en cas d'erreur.

Exemple : classe et objet

```php
// Définition d'une classe :
class Voiture
{
    private $couleur = "noire"; // attribut privé
    // constructeur
    function __construct($couleur="") {
        $this->couleur = $couleur;
    }
    // accesseur
    function getCouleur() {
        return $this->couleur;
    }
}

// Création d'une instance :
$monTacot = new Voiture("rouge");

echo "La couleur est ".$monTacot->getCouleur();
```

POO en PHP

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

☞ Lire le chapitre "Les classes et les objets" :
php.net/manual/fr/language.oop5.php

POO en PHP

- Dans une classe, le mot-clé `static` permet de définir des méthodes et des propriétés **statiques**.
- Le mot-clé `abstract` permet de définir des classes et des méthodes **abstraites**. Les classes définies comme abstraites ne peuvent pas être instanciées, et toute classe contenant au moins une méthode abstraite doit elle-aussi être abstraite.
- Les **interfaces** sont définies en utilisant le mot-clé `interface` (à la place de `class`), mais sans qu'aucune des méthodes n'ait son contenu de spécifié. Les interfaces spécifient quelles méthodes une classe doit implémenter, sans avoir à définir comment ces méthodes fonctionneront.
- Pour **implémenter une interface**, l'opérateur `implements` est utilisé (à la place de `extends`). Toutes les méthodes de l'interface doivent être implémentées dans une classe.

Les fonctions

➠ Une **fonction** peut être définie en utilisant la syntaxe suivante :

```php
function foo($arg_1, &$arg_2, $arg_3="toto")
{
    // $arg_1 => passage par valeur
    // &$arg_2 => passage par référence
    // $arg_3="toto" => valeur par défaut
    return $retval;
}
```

- Toutes les fonctions et classes en PHP ont une portée globale
- PHP ne supporte pas la surchage, la destruction et la redéfinition de fonctions déjà déclarées
- PHP supporte le passage d'arguments par valeur (comportement par défaut), le passage par référence, et des valeurs d'arguments par défaut

Exemple : une fonction

```php
<?php

function calculerMoyenne($notes)
{
    $total = 0;
    for($i = 0; $i < count($notes); $i++)
    {
        $total += $notes[$i];
    }
    return ($total/count($notes));
}

$notes = array(10, 12.5, 15, 17.5, 20);

$moyenne = calculerMoyenne($notes);

echo $moyenne;

?>
```

Variables prédéfinies

➠ PHP fournit un très grand nombre de variables prédéfinies accessibles à tous les scripts. Ces variables représentent à peu près tout, allant des variables externes aux variables d'environnement intégrées à PHP, en passant par les derniers messages d'erreur ou les en-têtes récupérés.
➠ Les variables **Superglobales** sont des variables internes qui sont toujours disponibles, quel que soit le contexte : `$_SERVER`, `$_GET`, `$_POST`, `$_FILES`, `$_REQUEST`, `$_SESSION`, `$_ENV` et `$_COOKIE`.
➠ Le tableau associatif `$GLOBALS` contenant les références sur toutes les **variables globales** actuellement définies dans le contexte d'exécution global du script. Les noms des variables sont les index du tableau.
➠ Les variables `$php_errormsg` (le dernier message d'erreur), `$http_response_header` (en-têtes de réponse HTTP), `$argc` et `$argv` (le nombre et leableau d'arguments passés au script).

Les variables Superglobales

- `$_SERVER` : Variables de serveur et d'exécution
- `$_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

Exemple : utilisation d'une variable Superglobale

```php
<?php

echo $_SERVER['HTTP_USER_AGENT'];

if(stripos($_SERVER['HTTP_USER_AGENT'], 'linux') !== FALSE)
{
    echo '<br />Vous utilisez un système d\'exploitation Linux<br />';
}
else if(stripos($_SERVER['HTTP_USER_AGENT'], 'windows') !== FALSE)
{
    echo '<br />Vous utilisez un système d\'exploitation Windows<br />';
}

?>
```

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
- `TRUE` (ou `true`) et `FALSE` (ou `false`) : les valeurs booléenne vraie et faux

Mes constantes

➠ Le programmeur peut lui aussi définir ses propres constantes en utilisant `define()` :

```php
<?php
        define("MA_CONSTANTE", "Hello world !");
        echo MA_CONSTANTE; // affiche Hello world !
?>
```

Les constantes ne commencent pas par le signe '$'. Elles sont accessibles partout, de manière globale et ne peuvent pas être redéfinies, ou indéfinies, une fois qu'elles ont été définies. Les constantes ne représentent que des valeurs scalaires : il n'est pas possible de définir des tableaux ou des objets.

Arrêt d'exécution d'un script

- L'exécution d'un script ne s'arrête que dans le cas suivants :
  - Lorsque celui-ci est terminé
  - Lorsque le temps d'exécution du script dépasse la limite autorisée par le serveur (paramètre `max_execution_time` de `php.ini`)
  - Lorsque le *parser* rencontre une erreur (voir traitement des erreurs)
  - Par un appel à `die()` ou `exit()` :

```php
<?php

...

// die arrête un script et affiche un message d'erreur dans le navigateur
// exit l'arrête aussi mais sans afficher de message d'erreur

if($etat == false)
  die("Erreur !");

?>
```

L'affichage

➠ PHP fournit 3 fonctions permettant d'envoyer du texte au navigateur : `echo`, `print` et `printf`.

➠ Pour améliorer la gestion de l'affichage, il existe plusieurs approches :

- concaténer l'intégralité du code HTML dans une variable et de réaliser un simple `echo` de cette variable en fin de script.
- utiliser les fonctions de bufferisation de sortie (`ob`) pour contrôler quand les données sont envoyées par le script.
- utiliser l'extension DOM.
- séparer la présentation (HTML) du contenu (PHP) en utilisant les *templates* (cf. Bonne pratique n°7).

Exemple : un simple echo

```php
// on initialise une variable de sortie
$out = "<html>\n";

// On concatène avec l'opérateur . en PHP
$out .= "\t<head><title>Exercice PHP</title></head>\n";
$out .= "\t<body><p>Du code HTML généré par PHP</p></body>\n";
$out .= "</html>";

// On affiche le contenu de la variable et donc de la page
echo $out;
```

Exemple : la bufferisation de sortie

```php
// démarre la temporisation de sortie
ob_start();

// l'instruction echo est stockée dans un buffer jusqu'à l'appel de la fonction
    ob_end_flush()
echo "Du code HTML généré par PHP\n";

// envoie les données du tampon de sortie et met fin à la temporisation de sortie
ob_end_flush();
```

Exemple : l'objet DOMDocument

```php
// Crée un nouvel objet DOMDocument
$doc = new DOMDocument('1.0');

$root = $doc->createElement('html');
$root = $doc->appendChild($root);
$head = $doc->createElement('head');
$head = $root->appendChild($head);
$body = $doc->createElement('body');
$body = $root->appendChild($body);
$p = $doc->createElement('p');
$p = $body->appendChild($p);
$text = $doc->createTextNode('Ceci est un paragraphe');
$text = $p->appendChild($text);

// sauvegarde le document interne dans une chaîne en utilisant un formatage HTML
    et l'affiche avec echo
echo $doc->saveHTML();
```

Sommaire

Gérer les données I

- Base de données
  - Avantage(s) : langage SQL
  - Inconvénient(s) : lenteur
- Fichier
  - Avantage(s) : rapidité
  - Inconvénient(s) : droits d'accès, structure
- Cookies
  - Avantage(s) : simplicité
  - Inconvénient(s) : dépendant du client, sécurité
- Sessions
  - Avantage(s) : sécurité
  - Inconvénient(s) : dépendant du serveur, complexité

Gérer les données II

- Formulaire
  - Avantage(s) : saisie possible
  - Inconvénient(s) : sécurité, dépendant de l'envoi
- URL
  - Avantage(s) : simplicité
  - Inconvénient(s) : visibilité, sécurité, taille des données, pas de saisie, dépendant d'hyperliens

Formulaire

➠ Un simple formulaire HTML :

```
<form action="action.php" method="post">
 <p>Votre nom : <input type="text" name="nom" /></p>
 <p>Votre âge : <input type="text" name="age" /></p>
 <p><input type="submit" value="OK"></p>
</form>
```

➠ Lorsque le visiteur remplit le formulaire, et clique sur le bouton OK, le script `action.php` est appelé :

```
echo "<pre>"; var_dump($_POST); echo "</pre>";
```

➠ Les données sont récupérées automatiquement dans le tableau Superglobal `$_POST` : `$_POST['nom']` et `$_POST['age']` ici

URL et hyperlien

➠ Un simple lien en HTML :

```
<a href="page.php?id=1">page 1</a>
```

➠ Lorsque le visiteur clique sur le lien, le script `page.php` est appelé :

```
echo "<pre>"; var_dump($_GET); echo "</pre>";
```

➠ Les données sont récupérées automatiquement dans le tableau Superglobal `$_GET` : `$_GET['id']` ici

Redirection

➠ La fonction `header()` permet l'envoi d'en-têtes personnalisés (<HEAD>). Le rôle des entêtes est d'échanger des méta-informations entre le serveur et le client. Ici, on peut utiliser un en-tête `Location` pour provoquer une redirection :

```php
<?php
     header("Location: page.php?id=2");
?>
```

➠ Les données sont récupérées automatiquement dans le tableau Superglobal `$_GET` : `$_GET['id']` ici

Les entêtes doivent obligatoirement être envoyées avant l'affichage de tout caractère dans la page en cours. Car l'affichage, se situant dans le `BODY` de la page HTML, force l'envoi des entêtes `HEAD` de base. Les entêtes peuvent servir à la redirection, à l'authentification, à l'envoi d'images ....

Manipulation de fichiers I

➠ PHP fournit de nombreuses fonctions pour manipuler des fichiers d'un système de fichiers :
fr.php.net/manual/fr/book.filesystem.php.

- Les appels de base pour la gestion des E/S fichiers sont : `fopen`, `fread`, `fwrite`, `fclose`, ...
- L'association entre une ressource nommée et un nom physique s'effectue à l'ouverture du fichier. La ressource nommée représentant le flux est en fait un pointeur de fichier.
- Le nom physique du fichier est une chaîne de caractères contenant son nom et éventuellement son chemin dans l'arborescence du système de fichiers géré par l'OS.
- L'ouverture d'un fichier se fait suivant un mode qui spécifie le type d'accès désiré au flux (lecture seule, écriture seule, lecture/écriture, ...).

Manipulation de fichiers II

- Les fonctions de lecture `fgetc()`, `fgetcsv()`, `fgets()`, `fscanf()` et `fgetss()` peuvent être intéressantes pour réaliser certains traitements spécifiques.
- Certaines fonctions de lecture n'ont pas besoin de réaliser d'ouverture préalable du fichier à traiter. C'est le cas de :
  - **`file()`** qui lit le fichier et renvoie le résultat dans un tableau
  - `file_get_contents()` qui lit tout un fichier dans une chaîne
- Pour écrire dans un fichier, on pourra utiliser au choix les fonctions **`fwrite()`**, `fputs()` ou même `file_put_contents()` et `fputcsv()` dans certains cas précis.

Exemple : Lire dans un fichier

```php
$filename = "datas.txt";
if (@file_exists($filename)) {
   $fichier = @fopen($filename, "r");
   if($fichier != FALSE)
   {
      $datas = fread($fichier, 10);
      echo "Dix premiers octets du fichier $filename : " . $datas . "<br />";
      $datas = fread($fichier, 10);
      echo "Dix octets suivants du fichier $filename : " . $datas . "<br />";
      fclose($fichier);
   }
   else die("Erreur : ouverture impossible du fichier $filename !<br />");
}
else die("Erreur : le fichier $filename n'existe pas !<br />");
```

Accès aux base de données I

➠ Parmi les nombreux atouts du langage PHP, un des plus connus est son interfaçage avec la majorité des bases de données du marché. Parmi les plus connues, on peut citer : **MySQL**, **SQLite**, PostgreSQL, Oracle, Ingres, Interbase, Informix, Microsoft SQL Server, mSQL, Sybase, FrontBase, dBase, etc ...

➠ La base de donnée la plus utilisée avec PHP est sans aucun doute : MySQL, un SGDBR (Système de Gestion de Base de Données Relationnelle) GPL implémentant le langage de requête SQL (*Structured Query Language*). Avec MySQL vous pouvez créer plusieurs bases de données sur un serveur. Une base est composée de tables contenant des enregistrements.

Il existe un outil libre et gratuit développé en PHP par la communauté des programmeurs libres : **phpMyAdmin**, qui permet l'administration aisée des bases de données MySQL avec PHP.

Accès aux base de données II

- PHP offre 3 APIs différentes pour se connecter à MySQL :
  - **mysql** : ancienne extension devenue obsolète depuis PHP 5.5 et sera supprimée dans un futur proche.
  - **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.

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

Accès aux base de données III

- 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`)

➠ L'exécution d'une requête `SELECT` avec `mysqli_query()` retournera 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()` : récupère une ligne de résultat sous forme de tableau indexé
- `mysqli_fetch_array()` : retourne une ligne de résultat sous la forme d'un tableau associatif, d'un tableau indexé, ou les deux

Accès aux base de données IV

- `mysqli_fetch_assoc()` : récupère une ligne de résultat sous forme de tableau associatif
- `mysqli_fetch_object()` : retourne la ligne courante d'un jeu de résultat sous forme d'objet
- et `mysqli_free_result()` : libère la mémoire associée à un résultat

Exemple : mysqli interface par fonctions

```php
// mysqli : http://fr.php.net/manual/fr/book.mysqli.php
if (!extension_loaded('mysqli'))
   die("L'extension mysqli n'est pas présente !");

$link = mysqli_connect('localhost', 'root', 'password', 'test');
if (!$link)
   die('Echec de connexion au serveur de base de données : ' .
       mysqli_connect_error() . '(' . mysqli_connect_errno() . ') ');

echo 'Fonctions mysqli : succès ... ' . mysqli_get_host_info($link) . " - MySQL
    server version : " . mysqli_get_server_info($link) . "<br />\n";

mysqli_close($link);
```

Exemple : mysqli interface orientée objet

```php
if (!class_exists('mysqli'))
   die("La classe mysqli n'est pas présente !");
// ou :
if(!in_array("mysqli", get_declared_classes()))
   die("La classe mysqli n'est pas présente !");

$mysqli = new mysqli("localhost", "root", "password", "test");
if ($mysqli->connect_error)
    die('Echec de connexion au serveur de base de données : ' . $mysqli->
        connect_error . '(' . $mysqli->connect_errno . ') ');

echo 'Classe mysqli : succès ... ' . $mysqli->host_info . " - MySQL server
    version : " . $mysqli->server_info . "<br />\n";

$mysqli->close();
```

Exemple : PDO

```php
// PDO : http://www.php.net/manual/en/book.pdo.php
// pdo_mysql : http://www.php.net/manual/en/ref.pdo-mysql.php

if(!in_array("PDO", get_loaded_extensions()))
   die("L'extension PDO n'est pas présente !<br><br>");

if(!in_array("pdo_mysql", get_loaded_extensions()))
   die("L'extension pdo_mysql n'est pas présente !<br><br>");

$pdo_db = new PDO('mysql:host=localhost;dbname=test', 'root', 'password') or die
     ("Echec de la création de l'instance PDO !");

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

unset($pdo_db);
```

Exemple : mysqli interface par fonctions

```
if ($result = mysqli_query($link, "SELECT Host, User FROM ‘user‘ ORDER BY User
    DESC LIMIT 0 , 30")) {
   printf("Fonctions mysqli : la requête a retourné %d enregistrement(s).<br />\
       n", mysqli_num_rows($result));
   /* Tableau indexé */
   //$row = mysqli_fetch_array($result, MYSQLI_NUM);
   //while(list($host, $user) = mysqli_fetch_row($result)) {
   //   echo "$host - $user<br />";
   //}
   /* Tableau associatif */
   //$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
   while($row = mysqli_fetch_array($result)) {
      $host = $row["Host"];
      $user = $row["User"];
      echo "$host - $user<br />";
   }
   /* Libération des résultats */
   mysqli_free_result($result);
}
```

Exemple : mysqli interface orientée objet

```php
if ($result = $mysqli->query("SELECT Host, User FROM ‘user‘ ORDER BY User DESC
     LIMIT 0 , 30"))
{
   printf("Classe mysqli : la requête a retourné %d enregistrement(s).<br />\n",
         $result->num_rows);
   /* Tableau indexé */
   //$row = $result->fetch_array(MYSQLI_NUM);
   //while(list($host, $user) = $result->fetch_array()) {
   //   echo "$host - $user<br />";
   //}
   /* Tableau associatif */
   //$row = $result->fetch_array(MYSQLI_ASSOC);
   while($row = $result->fetch_array()) {
      $host = $row["Host"];
      $user = $row["User"];
      echo "$host - $user<br />";
   }
   /* Libération des résultats */
   $result->free();
}
```

Exemple : PDO

```
if ($result = $pdo_db->query("SELECT Host, User FROM ‘user‘ ORDER BY User DESC
     LIMIT 0 , 30"))
{
   printf("Classe PDO : la requête a retourné %d enregistrement(s).<br />\n",
        $result->rowCount());
   /* Tableau indexé */
   //$row = $result->fetch(PDO::FETCH_NUM);
   //printf ("%s - %s<br />\n", $row[0], $row[1]);

   /* Tableau associatif */
   //$row = $result->fetch(PDO::FETCH_ASSOC);
   //printf ("%s - %s<br />\n", $row["User"], $row["Host"]);

   // ou tous les résultats de la requête :
   $datas = $result->fetchAll();
   echo "<pre>"; print_r ($datas); echo "</pre>";
}
```

Sommaire

Bonne pratique n°1 : traitement des erreurs

➠ Dans `php.ini`, les directives suivantes sont nécessaires en développement mais ne doivent jamais être utilisées sur un système en production (un système connecté à Internet) :

- La directive `display_errors` doit être fixée à `On` pour afficher les erreurs à l'écran
- La directive `error_reporting` qui permet de choisir le niveau d'erreur doit être mise à `E_ALL` pour détecter toutes les erreurs

☞ *Remarque* : l'@ placé devant une fonction bloquera les messages d'erreurs envoyés au navigateur client

Il est recommandé d'utiliser l'historique d'erreur (directive `log_errors`), plutôt que d'afficher les erreurs (directive `display_errors`) sur les sites de production.

Bonne pratique n°2 : structure des scripts

➠ En partant du principe que la page PHP demandée est équivalente à un "*main*" :

- Cette page pourra utiliser des fonctions ou des méthodes de classes.
- On regroupera ces fonctions et classes dans des fichiers externes.
- Pour ces fichiers, on utilisera par convention les extensions suivantes : `.inc.php` pour des fonctions et `.class.php` pour des classes
- Si la page en a besoin, il lui faudra préalablement inclure ces fichiers en utilisant : `include()` ou `require()`

Il est recommandé d'utiliser ce principe pour s'assurer que les scripts qui ne contiennent que des fonctions et des classes ne "produiront" rien si ils sont appelés directement par l'internaute. En effet, ils ne contiendront aucune instruction exécutée directement et aucun détournement de script ne sera alors possible.

Bonne pratique n°3 : les variables globales

➠ Comme dans tout langage de programmation, il est déconseillé d'utiliser les variables globales :

- celles du programme : préférer le passage de paramètres aux fonctions
- celles de configuration : utiliser des tableaux associatifs ou des constantes
- celles anciennement prédéfinies automatiquement par PHP (`$HTTP_xxx_VARS`) : mettre `register_globals = Off` dans `php.ini` et utiliser les variables Superglobales

Bonne pratique n°4 : vérifier les dépendances

➠ Pour améliorer la robustesse du script, il est conseillé de vérifier ses dépendances :

- vis à vis des options de `php.ini` :
  - La fonction `ini_get(string varname)` retourne la valeur de l'option de configuration `varname` en cas de succès, sinon `FALSE`.
  - la fonction `get_cfg_var(string varname)` retourne la valeur courante de l'option PHP `varname`.
- vis à vis d'un module ou d'une fonction PHP :
  - La fonction `get_loaded_extensions()` retourne un tableau contenant les noms de tous les modules compilés et chargés sur l'interpreteur PHP courant.
  - la fonction `get_defined_functions()` retourne un tableau multi-dimensionnel, contenant la liste de toutes les fonctions définies, aussi bien les fonctions internes à PHP que celle déjà définie par l'utilisateur.

Bonne pratique n°5 : vérifier les données I

➠ La plus grande faiblesse de nombreux programmes PHP ne vient pas du langage en lui-même, mais de son utilisation en omettant les caractéristiques de sécurité. Il faut donc porter une attention particulière aux variables (provenance, type, initialisation, ...) car elles sont susceptibles de fausser le comportement d'un script. On peut utiliser :

- `Empty()` : détermine si une variable contient une valeur non nulle.
- `IsSet()` : détermine si une variable est définie et non nulle.
- `defined()` : vérifie si une constante est définie.
- `is_null()`, `is_bool()`, `is_float()` ou `is_int()` : détermine si une variable vaut `NULL`, si une variable est un booléen, un nombre décimal ou un nombre entier.
- `is_numeric()` ou `is_string()` : détermine si une variable est un type numérique ou de type chaîne de caractères.

Bonne pratique n°5 : vérifier les données II

- `is_a()` ou `is_subclass_of()` : vérifie si l'objet est une instance d'une classe donnée ou si un objet est une sous-classe d'une classe donnée
- `===` : vérifie le type d'une variable dans un test conditionnel

Bonne pratique n°6 : nettoyage I

➠ Pour des raisons de sécurité évidente, il faut porter une attention particulière à toute donnée extérieure (en provenance d'URL, de formulaire, ...) susceptible d'injecter du code (SQL, Javascript, PHP, ...) suivant des méthodes nommées XSS/CSS (*Cross Site Scripting*).

- `htmlspecialchars()` : convertit tous les caractères spéciaux en entité HTML.
- `htmlentities()` : convertit tous les caractères spéciaux en entité HTML.
- `strip_tags()` : enlève les balises HTML et PHP.
- `AddSlashes()` : ajoute un *slash* devant tous les caractères spéciaux (et `StripSlashes()` les enlève).
- `eregi()` : recherche par expression rationnelle insensible à la casse

Bonne pratique n°6 : nettoyage II

- `preg_replace()` : rechercher et remplace par expression rationnelle standard
- `urlencode()` et `urldecode()` : encode une chaîne en URL et décode une chaîne encodée URL

Bonne pratique n°7 : les *templates* I

- Ils permettent de séparer la présentation (HTML) du contenu (PHP).
  - Il faut éviter au maximum de mélanger les couches (architecture n-tiers). La meilleure utilisation est dans une situation où le(s) codeur(s) et le(s) designer(s) ne sont pas la même personne.
  - Avec les *templates*, les designers ont un contrôle total sur la présentation de l'application en modifiant simplement des fichiers *templates* et les développeurs ne se concentreront que sur l'application et le passage de variables aux fichiers *templates* associées.
  - D'un côté, nous aurons le programme, et de l'autre un document HTML, le moteur de template se chargeant de réaliser le mixage entre les deux, grâce à un type de balisage sécifique au template (modèle).

Bonne pratique n°7 : les *templates* II

- Ce type de solution a fait ses preuves en PHP à travers des moteurs du type de **Smarty**, PHPlib, FastTemplates ou ModeliXe. Le framework PEAR (*PHP Extension and Application Repository*) propose aussi plusieurs solutions de templates dont HTML_Template_Flexy.

Exemple : Smarty

Le fichier index.php

```php
<?php
require("Smarty.class.php");
$smarty = new Smarty;
$smarty->assign("var","world");
$smarty->display("index.tpl");
?>
```

Le fichier templates/index.tpl

```
<HTML>
<TITLE>Premier test</TITLE>
<BODY>
Hello {$var} !
</BODY>
</HTML>
```

Les *frameworks* PHP I

➠ Un *framework* est un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture).

- fr.wikipedia.org/wiki/Liste_de_frameworks_PHP
- Parmi les plus connus et utilisés : Zend Framework, **Symfony**, CakePHP, ...
- Beaucoup de ces *frameworks* utilisent le modèle MVC (Modèle-Vue-Contrôleur) :
  - un modèle (modèle de données) ;
  - une vue (présentation, interface utilisateur) ;
  - un contrôleur (logique de contrôle, gestion des événements, synchronisation).

Les *frameworks* PHP II

- Exemple : le modèle MVC (Modèle-Vue-Contrôleur) selon TinyMVC