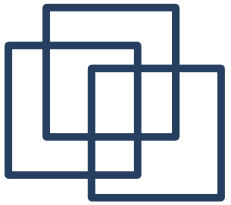
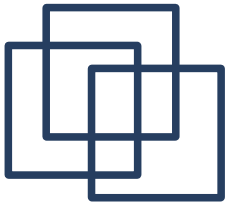


PROGRAMMATION PHP

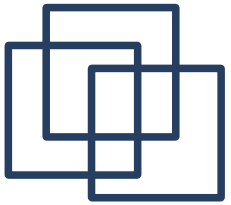
1. Serveur web Apache
2. PHP CLI
3. Ressources
4. Arrêt d'exécution d'un script
5. Structure d'un script
6. Inclusion
7. Conserver des données
8. Passer de page en page
9. Variables d'environnement
10. Variables globales
11. Constantes
12. php.ini
13. Traitement d'erreurs
14. Dépendances d'un script
15. Corruption de variables
16. Nettoyer du code HTML/PHP
17. Les templates
18. Convention de codage



1 . Serveur Web Apache (1)

Apache HTTP Server, souvent appelé Apache, est un logiciel de serveur HTTP produit par l'Apache Software Foundation. C'est le serveur HTTP le plus populaire du Web. C'est un logiciel libre avec un type spécifique de licence, nommée licence Apache.

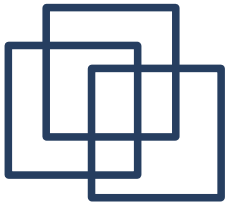
Apache est conçu pour prendre en charge de nombreux modules lui donnant des fonctionnalités supplémentaires : interprétation du langage Perl, PHP, Python et Ruby, serveur proxy, CGI, réécriture d'URL, etc. Néanmoins, il est à noter que l'existence de nombreux modules Apache complexifie la configuration du serveur web. En effet, les bonnes pratiques recommandent de ne charger que les modules utiles : de nombreuses failles de sécurité affectant uniquement les modules d'Apache sont régulièrement découvertes.



1 . Serveur Web Apache (2)

Les possibilités de configuration d'Apache sont une fonctionnalité phare. Le principe repose sur une hiérarchie de fichiers de configuration, qui peuvent être gérés indépendamment. Cette caractéristique est notamment utile aux hébergeurs qui peuvent ainsi servir les sites de plusieurs clients à l'aide d'un seul serveur HTTP. Pour les clients, cette fonctionnalité est rendue visible par le fichier .htaccess.

Parmi les logiciels aidant la maintenance d'Apache, les fichiers de log peuvent s'analyser à l'aide de nombreux scripts et logiciels libres tels que AWStats, Webalizer ou W3Perl. Plusieurs interfaces graphiques facilitent la configuration du serveur.

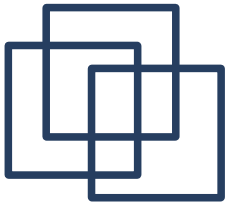


1 . Serveur Web Apache (3)

Le nom donné au fichier de configuration varie selon les distributions. Il est généralement désigné par httpd.conf (sous Fedora, Redhat, Mandriva, etc.) ou par apache2.conf (sous Debian, etc.). Il est situé dans le répertoire /etc/httpd/conf ou /etc/apache2/conf, selon les distributions.

La configuration d'Apache prend en compte beaucoup d'éléments qui peuvent être regroupés en trois grandes sections (voir annexe) :

- Environnement global
- Configuration du serveur
- Hôtes virtuels



1 . Serveur Web Apache (4)

Sur une Mandriva, le service httpd peut être géré manuellement par :

- La commande `service` ou les scripts de `/etc/init.d/`

```
# service httpd
I need an action
Usage: /etc/init.d/httpd {start|stop|restart|reload|graceful|condreload|
closelogs|update|condrestart|status|extendedstatus|configtest|
configtest_vhosts|debug|show_defines}
```

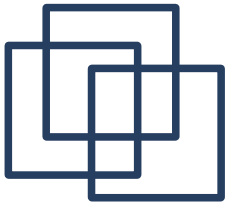
- Ou directement par la commande `apachectl`

Exemple : démarrer le serveur Apache manuellement (au choix)

```
# service httpd start
# /etc/init.d/httpd start
# apachectl start
```

Sur une Mandriva, le service httpd peut être géré automatiquement par la commande `chkconfig` :

```
# chkconfig --list httpd
```



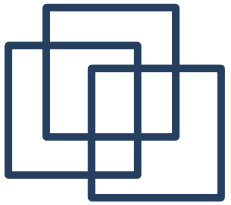
1 . Serveur Web Apache (5)

Les processus httpd (serveur web Apache) sont exécutés sous un compte défini dans le fichier de configuration du serveur.

Sous Linux :

```
# cat /etc/httpd/conf/httpd.conf | grep '\(User\|Group\)\+'
User apache
Group apache
```

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10655 apache    20   0 27876 3828 2364 S  0.0  0.4   0:00.00 httpd
10654 apache    20   0 27876 3588 2144 S  0.0  0.3   0:00.00 httpd
10652 apache    20   0 27876 3768 2236 S  0.0  0.4   0:00.00 httpd
 4143 apache    20   0 27876 3828 2376 S  0.0  0.4   0:00.00 httpd
 4142 apache    20   0 27876 1808 1252 S  0.0  0.2   0:00.00 httpd
 4141 apache    20   0 27876 3820 2368 S  0.0  0.4   0:00.00 httpd
 4140 apache    20   0 27876 3828 2376 S  0.0  0.4   0:00.00 httpd
 4139 apache    20   0 27876 1996 1340 S  0.0  0.2   0:00.00 httpd
 4138 apache    20   0 27876 1996 1340 S  0.0  0.2   0:00.00 httpd
 4137 apache    20   0 27876 3540 2132 S  0.0  0.3   0:00.03 httpd
 4136 apache    20   0 27876 1984 1324 S  0.0  0.2   0:00.02 httpd
 4121 root      20   0 27876  844  784 S  0.0  0.1   0:02.88 httpd
```

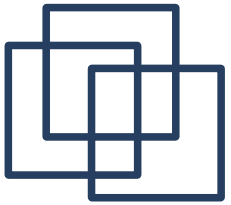


1 . Serveur Web Apache (6)

L’accès au(x) document(s) demandé par le client se fait sous le compte utilisé par le processus httpd sur le serveur hôte. Les droits d'accès sur le système de fichiers du serveur hôte s'applique :

```
# ls -l
-rw-r--r-- 1 tv     tv     3278 2009-09-11 11:51 bd_mediane.php
drwxr-xr-x 2 tv     tv     4096 2009-09-22 09:21 fichiers_mesure/
-rw-r----- 1 tv     apache 2309 2009-09-11 11:31 f_mediane.php
drwxrwx--- 2 apache apache 4096 2009-09-22 09:20 fichiers_mediane/
```

Si les permissions ne permettent pas d'accéder à la ressource, le serveur renverra une erreur 403.



2 . PHP CLI

Indépendemment du développement web, le langage PHP peut être utilisé aussi comme un langage de script en mode commande : *Command Line Interface* (CLI).

Exemple 1 : un script path.php

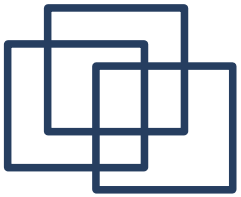
```
<?php
$path = getenv("PATH"); // retourne la variable PATH
echo "PATH = $path\n";
?>
```

Exécution : $ php -f path.php

Exemple 2 : un script path.php exécutable (avec le droit x)

```
#!/usr/bin/php
<?php
$path = getenv("PATH"); // retourne la variable PATH
echo "PATH = $path\n";
?>
```

Exécution : $ ./path.php

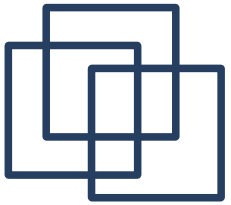


3 . Ressources

- Scripts : http://www.phpscripts-fr.net/
- Classes : http://www.phpclasses.org/
- Framework PEAR : http://pear.php.net/packages.php
- *Remarque:*

PEAR est un framework officiel et un système de distribution de composants réutilisables pour PHP. PEAR a été intégré dans le package de la release 4.3 de PHP. Il est donc considéré comme stable et utilisable dans un environnement professionnel.

Une présentation rapide en français : www.phpteam.net



4 . Arrêt d'exécution d'un script

L'exécution d'un script ne s'arrête que dans le cas suivants :

- Lorsque celui-ci est terminé
- Par un appel à `die()` ou `exit()` :
  - **`die`** arrête un script et affiche un message d'erreur dans le navigateur.

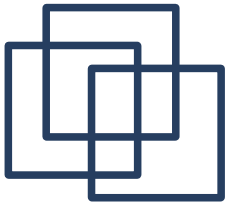
  Exemple :

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

  - **`exit`** l'arrête aussi mais sans afficher de message d'erreur.
- Lorsque le temps d'exécution du script dépasse la limite autorisée par le serveur (paramètre de `php.ini`) :

```
max_execution_time = 30     ; Maximum execution time of each script, in seconds
```

- Lorsque le *parser* rencontre une erreur (voir traitement des erreurs)



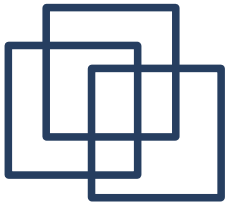
5 . Structure d'un script

- On peut comparer la page php demandée à un « *main* ».
- Cette page va utiliser des fonctions ou des classes qui sont définies et déclarées dans d'autres fichiers.
- Pour ces fichiers, on utilisera par convention les extensions suivantes :
  `.inc.php` pour des fonctions
  `.class.php` pour des classes
- Si la page en a besoin, il lui faudra inclure ces fichiers en utilisant :
  `include()` ou **`require()`**

```
// fichier : accueil.php
<?php
include("compteur.inc.php");
$count = Compter("accueil");
echo "Cette page a été visitée : ".$count;
…
?>
```

inclusion

```
// fichier compteur.inc.php
<?php
function Compter($page)
{
…
}
?>
```



6 . Inclusion

- **`include()`**

La fonction include() inclut et évalue le fichier spécifié en argument.
include() diffère de require() car le fichier inclus est ré-évalué à chaque fois que la commande est exécutée, tandis que require() est remplacée par le fichier cible lors de la première exécution, que son contenu soit utilisé ou non. De plus, cela se fait même s'il est placé dans une structure conditionnelle, comme dans un if(). Donc, vous pouvez utiliser la fonction include() dans une boucle pour inclure un nombre infini de fois un fichier, ou même des fichiers différents.

- **`require()`**

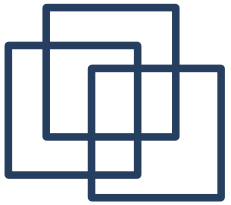
La commande require() se remplace elle-même par le contenu du fichier spécifié, comme les préprocesseurs C le font avec la commande #include.
require() n'est pas vraiment une fonction PHP : c'est plus une instruction du langage. Elle ne fonctionne pas comme les fonctions standards. Par exemple, require() est indépendante des structures de contrôle (cela ne sert à rien de la placer dans une condition ou une boucle, elle sera toujours exécutée et qu'une fois).

- **`include_once()`**

La commande include_once() inclut et évalue le fichier spécifié durant l'exécution du script et, si le code a déjà été inclus, il ne le sera pas une seconde fois.

- **`require_once()`**

La commande require_once() se remplace elle-même par le fichier spécifié et ce code ne sera ajouté qu'une seule fois, évitant de ce fait les redéfinitions de variables ou de fonctions, génératrices d'alertes.



7 . Conserver des données

On pourra récupérer des données de manière différente :

- dans une base de données

Avantage : langage SQL
Inconvénient : lenteur

- dans des fichiers sur le serveur

Avantage : plus rapide qu'une base de données
Inconvénient : oblige à donner des droits d'écriture aux visiteurs

- dans des *cookies*

Avantage : simple (très utilisé)
Inconvénients : le client peut refuser les *cookies* (stockage côté client et donc modifiables)

- dans l'URL

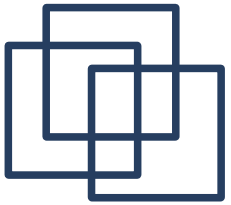
Avantage : très simple et permet l'interaction avec le client (choix)
Inconvénients : visibles dans la barre d'adresse et donc modifiables manuellement, impose de toujours utiliser les liens hypertextes

- dans un formulaire

Avantage : très simple et permet l'interaction avec le client (choix et saisie)
Inconvénients : impose l'envoi du formulaire, données modifiables (pour les champs cachés ou lecture seule)

- en utilisant le mécanisme des sessions (stockage côté serveur)

Avantage : plus sur (stockage côté serveur)
Inconvénients : gérer l'identifiant de session et le serveur doit accepter les sessions



8 . Passer de page en page

Sans tenir compte de la gestion des données, il est possible de passer de page en page de trois manières différentes :

- Par l'envoi d'un formulaire

```
<FORM name=formulaire action=page.php method=post>
…
</FORM>
```

- Par navigation sur les liens hypertextes

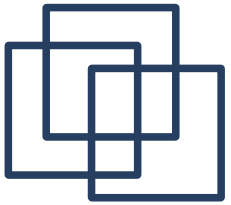
```
<a href=http://www.serveur.com/page.php?nom=… >lien</a>
```

- Par redirection en utilisant les entêtes

Le rôle des entêtes est d'échanger des méta-informations entre le serveur et le client.
La commande **`header()`** du php permet l'envoi d'entêtes personnalisées. Par exemple :

```
header("Location: page.php?nom=… "); //pour rediriger vers la page "page.php"
```

Les entêtes doivent obligatoirement être envoyées avant l'affichage de tout caractère dans la page en cours. Car l'affichage force l'envoi des entêtes de base.
Les entêtes peuvent servir à la redirection, à l'authentification, à l'envoi d'images ...
On peut les utiliser en HTML entre les balises **`<HEAD>`**…**`</HEAD>`**



9 . Variables d'environnement

Le langage php est doté d'une multitude de variables d'environnement que la fonction **phpinfo()** permet d'afficher (plus d'autres informations sur la configuration du serveur Apache).

Ces variables permettent au script d'accéder à des informations très utiles voire quelques fois indispensables.

Quelques variables :

$PHP_SELF : nom du script en cours

$HTTP_ACCEPT : liste des types MIME supportés par le client

$HTTP_USER_AGENT : signature du navigateur du client

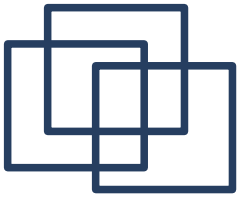
$REMOTE_ADDR : adresse IP du client

$QUERY_STRING : chaîne au format URL contenant les paramètres passés à la page en cours

$HTTP_REFERER : URL de la source ayant renvoyée le client sur la page en cours (en l'analysant, on peut connaître le moteur de recherche utilisé ainsi que les mots clés entrés par l'internaute, s'il vient effectivement d'un moteur de recherche)

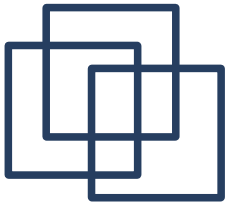
Ce mécanisme simple est rendu possible grâce au paramètre **`register_globals`** du fichier **`php.ini`**. Jusqu'à aujourd'hui ce paramètre est positionné à **On** par défaut (même avec la version PHP 4.1.0), ce qui permet de manipuler directement les variables globales de type ENV/GET/POST/COOKIE/SERVER (EGPCS).

De plus si vous avez positionné le paramètre **`track_vars`** du fichier **`php.ini`** à **On** (valeur par défaut depuis PHP 4.0.3), vous récupérez automatiquement les variables d'environnement, les variables serveur, et les variables GET/POST/Cookie dans les tableaux associatifs **$HTTP_ENV_VARS**, **$HTTP_SERVER_VARS**, **$HTTP_GET_VARS**, **$HTTP_POST_VARS**, et **$HTTP_COOKIE_VARS**.



10 . Variables globales (1)

- Comme dans tout en langage de programmation, on se doit d'éviter d'utiliser les variables globales :
  - celles du programme => préférer le passage de paramètres aux fonctions
  - variables globales de configuration par exemple => utiliser des tableaux associatifs ou des constantes
  - celles anciennement prédéfinies par PHP ($HTTP_xxx_VARS) ou directement => mettre **`register_globals = Off`** et utiliser les variables super-globales
- Rechercher les variables préfixées par **`global`** dans les fonctions



10 . Variables globales (2)

Le projet PHP a décidé de modifier à partir de la version **4.1.0** le principe de fonctionnement des variables globales.

A terme les paramètres **register_globals** et **track_vars** doivent disparaître du fichier **php.ini**, ce qui revient à dire qu'ils prendront tous les deux la valeur *Off*.

Il ne sera donc plus question d'utiliser directement les variables globales, mais plutôt les nouveaux tableaux associatifs correspondants.

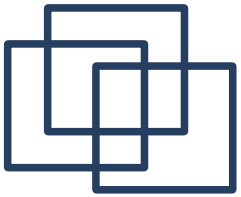
Pour faciliter la transition, PHP propose 3 nouvelles fonctionnalités :

- ces tableaux associatifs sont "créés" par défaut dans vos scripts sans qu'il soit nécessaire de mettre *track_vars* à *On*,
- ils sont accessibles à l'intérieur des fonctions sans qu'il soit nécessaire de les déclarer avec l'instruction `global`, = variables super globales
- ils changent de nom, pour devenir plus courts et plus simples. Désormais vous pourrez manipuler les tableaux **$_GET**, **$_POST**, **$_COOKIE**, **$_SERVER** et **$_ENV**. Les tableaux $HTTP_*_VARS correspondants sont amenés eux à disparaître.

L'objectif affiché par le PHP Group est que chaque administrateur de site positionne rapidement la variable register_globals à *Off*, et que chaque développeur PHP s'astreigne à manipuler les variables $_*.

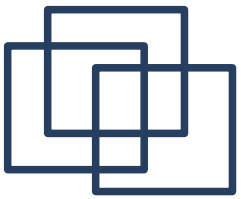
Pour les variables d'environnement, on privilégiera l'utilisation de la fonction `getenv()` :

```
$ip = getenv("REMOTE_ADDR"); // retourne l'adresse IP de l'utilisateur
```



10 . Variables super-globales (3)

- variables prédéfinies de PHP = variables globales du programme, accessibles de partout.
- Les variables super-globales sont les tableaux suivants :
  - $GLOBALS
  - $_SERVER (ancienne variable $HTTP_SERVER_VARS)
  - $_GET (ancienne variable $HTTP_GET_VARS)
  - $_POST (ancienne variable $HTTP_POST_VARS)
  - $_COOKIE (ancienne variable $HTTP_COOKIE_VARS)
  - $_FILES (ancienne variable $HTTP_POST_FILES)
  - $_ENV (ancienne variable $HTTP_ENV_VARS)
  - $_REQUEST
  - $_SERVER (ancienne variable $HTTP_SERVER_VARS)
  - $_SESSION (ancienne variable $HTTP_SESSION_VARS)



10 . Variables (4) : EGPCS

- les variables GET/POST/Cookie : utiliser **`import_request_variables()`** qui importe les variables GET/ POST/Cookie dans l'environnement global.

  bool import_request_variables (string types, string prefix)

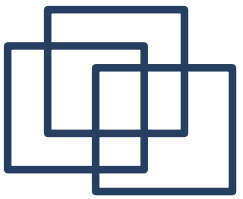
  - le paramètre types , permet de spécifier les variables à importer. 'G', 'P' et 'C' désignent respectivement GET, POST et Cookie. (POST inclus les fichiers uploadés)

  - le paramètre prefix est utilisé comme un préfixe de nom de variable, qui sera ajoutée au début de tous les noms de variables importées.

- Les autres variables : utiliser de la fonction **`extract()`**

  int extract(array var_array, int extract_type, string prefix)

  extract sert à exporter un tableau vers la table des symboles. Elle prend un tableau associatif var_array , crée les variables dont les noms sont les index de ce tableau, et leur affecte la valeur associée. Pour chaque paire clé/valeur, extract crée une variable, avec les paramètres extract_type et prefix .



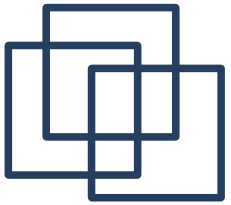
10 . Variables (5) : Astuce

- Utilisation d'anciens scripts avec du PHP sécurisé par **`register_globals = Off`**

- Il est peut-être difficile (long et fastidieux) d'assurer un portage d'un ancien script.

- Astuce à placer en début de chaque script:

```
// simuler le register_globals = On
foreach($_REQUEST as $a => $b)
{
   $$a = $b;
}
```



11 . Constantes

Le langage php met à disposition du programmeur des constantes propres au script qui ne peuvent pas être redéfinies.

Les constantes prédéfinies du PHP :

__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) : la valeur vraie booléenne

FALSE (ou false) : la valeur faux booléenne

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

```
define("CONSTANTE",  "Hello world !");
echo CONSTANTE; //affiche Hello world !
```

Remarques :

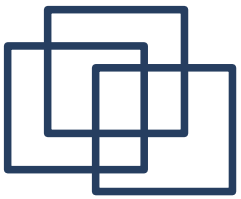
Les constantes ne commencent pas par le signe '$'

Les constantes sont accessibles partout, de manière globale.

Les constantes 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.

On utilisera, de préférence, les majuscules pour définir puis identifier une constante (règle de codage).



12 . php.ini

Les options de configuration du langage PHP sont définis dans un fichier php.ini. Il est recommandé de respecter les options suivantes :

- **register_globals = Off [sécurité, performance]**

Il sera de toute façon mis à Off par défaut (c'est à dire aussi dans le fichier php.ini-dist).

- **display_errors = Off et log_errors = On [sécurité]**

Plus de message d'erreurs à l'écran, pouvant révéler par exemple l'emplacement de vos fichiers inclus. Laisser à On sur la machine de développement, mais à Off sur celle en production. En complément du paramètre display_errors, log_errors permet de récupérer les messages d'erreurs dans les fichiers de *log* pour analyse, puisqu'ils ne sont plus visualisables à l'écran.

- **magic_quotes_gpc = Off [performance]**

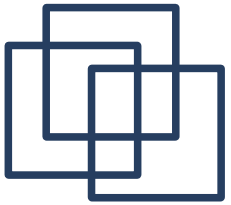
Plus backslashs automatique. Leur insertion dans une base de données devra se faire avec la fonction addslashes(). Peut obliger à modifier la totalité des scripts ...

- **variables_order = "GPCS" [performance]**

Les variables d'environnement ne sont plus disponibles dans le tableau $HTTP_ENV_VARS[]. Il faut utiliser la fonction getenv() pour y accéder.

- **error_reporting = E_ALL [code plus propre, sécurité(?)]**

Toutes les erreurs de scripts sont signalées, dont celles pouvant survenir lorsqu'une variable est utilisée sans être préalablement initialisée. (voir plus loin)



13. Traitement des erreurs

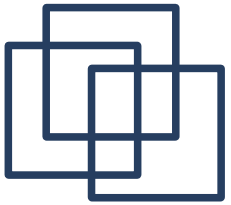
La fonction **`error_reporting($nbr)`** permet de fixer le niveau de rapport d'erreurs PHP.

Par défaut, php est assez permissif puisqu'il autorise l'utilisation des variables avant leur création, le cast implicite, l'appel de fonction retournant une valeur sans variable pour la récupérer…

Exemples :

```
error_reporting(E_ERROR | E_WARNING | E_PARSE);
//ou
error_reporting(E_NOTICE);
//ou
error_reporting(E_ALL);
```

constante	valeur
1	E_ERROR
2	E_WARNING
4	E_PARSE
8	E_NOTICE
16	E_CORE_ERROR
32	E_CORE_WARNING
64	E_COMPILE_ERROR
128	E_COMPILE_WARNING
256	E_USER_ERROR
512	E_USER_WARNING
1024	E_USER_NOTICE



13. Traitement des erreurs (2)

Les constantes suivantes, lorsqu'elles sont déclarées par le programmeur (en général avant toute les autres instructions du script), permettent de spécifier à l'interpréteur php du serveur quel niveau de rigueur appliquer face aux erreurs lors du déroulement du script.

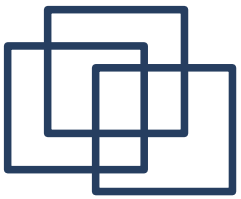
E_ERROR : dénote une erreur autre qu'une erreur d'analyse ("parse error") qu'il n'est pas possible de corriger.

E_WARNING : dénote un contexte dans lequel le PHP trouve que quelque chose ne va pas. Mais l'exécution se poursuit tout de même. Ces alertes-là peuvent être récupérées par le script lui-même.

E_PARSE : l'analyseur a rencontré une forme syntaxique invalide dans le script, correction de l'erreur impossible.

E_NOTICE : quelque chose s'est produit, qui peut être ou non une erreur. L'exécution continue. Par exemple, la tentative d'accéder à une variable qui n'est pas encore affectée.

E_ALL : toutes les constantes E_* rassemblées en une seule. Si vous l'utilisez avec **`error_reporting(),`** toutes les erreurs et les problèmes que PHP rencontrera seront notifiés.

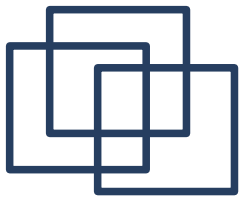


14 . Dépendances d'un script

- dépend d'une option de php.ini
  - La fonction **`ini_get(string varname)`** retourne la valeur de l'option de configuration varname en cas de succès, et FALSE.
  - Voir aussi: la fonction `get_cfg_var(string varname)` qui retourne la valeur courante de l'option PHP varname.
- dépend d'un module de PHP
  - La fonction **`get_loaded_extensions()`** retourne un tableau contenant les noms de tous les modules compilés et chargés sur l'interpreteur PHP courant. Elle permettra d'assurer un fonctionnement correct du script :

```
if(!in_array("xml", get_loaded_extensions()))
   die("Il manque l'option -with-xml");
```

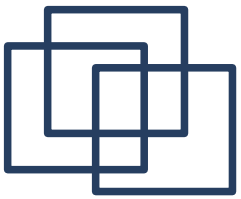
  - Voir aussi: la fonction `get_defined_functions()` retourne un tableau multi-dimensionnel, contenant la liste de toutes les fonctions définies, aussi bien les fonctions internes à PHP que celle déjà définie par l'utilisateur. Les noms des fonctions internes sont accessibles via $arr["internal"], et les fonctions utilisateur sont accessibles via $arr["user"]. On peut s'en servir pour vérifier la présence d'une fonction et assurer un fonctionnement correct du script.



15 . Corruption de variables (1)

Pour des raisons de sécurité évidente, il faut porter une attention particulière aux variables (provenance, type, initialisée, …) car elles sont susceptibles de fausser le comportement d'un script.

- IsSet() : Détermine si une variable est définie et est différente de NULL. Remarque : Pour vérifier si une constants est définie, utilisez la fonction defined().
- Empty() : Détermine si une variable contient une valeur non nulle.
- Les types (http://fr.php.net/manual/fr/language.types.php) : il est important de comprendre les types et leur signification. Par exemple, "42" est une chaîne de caractères, alors que 42 est un entier. FALSE est boolean alors que "false" est une chaîne de caractères. Remarque : Les formulaires HTML ne connaissent pas les entiers, nombres à virgules et autres booléens. Pour savoir si une structure est un entier, utilisez is_numeric().
- Comparaison de types (http://fr.php.net/manual/fr/types.comparisons.php) : préférez la comparaison stricte avec ===



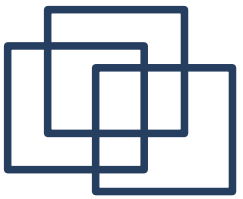
15 . Corruption de variables (2)

- provenance de la variable (GET, POST, COOKIE ?) => **`register_globals = Off`**
  - Exemple: la variable doit venir d'un formulaire par la "method" POST

```
if($_POST['expediteur'] && !$_COOKIE['expediteur'] && !$_GET['expediteur'] )
  {
   // maintenant, il faut vérifier le contenu de la variable
   $from = $_POST["expediteur"];
} else {
   mail("webmaster@nowhere.com", "Tentative de piratage",
  $_SERVER['REMOTE_ADDR']);
   echo "Problème de sécurité, l'administrateur est alerté.";
   exit;
}
```

- contenu de la variable => expressions régulières (dépend de la variable)
  - Exemple : injection d'*header* (cc, bcc, ...) dans la fonction `mail()`

```
if (eregi("\r", $from) || eregi("\n", $from)){
   die("Houston, on a un problème !");
}
```

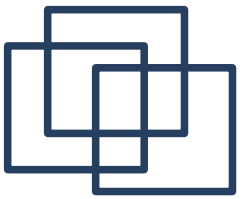


16 . Nettoyer du code HTML/PHP indésirable (1)

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).

- **`AddSlashes()`** : ajoute un slash devant tous les caractères spéciaux.
- **`StripSlashes()`** : enlève les slashs ajoutés par la fonction addslashes()
- *Remarque:* les guillemets magiques
  - Il existe une option PHP qui permet d'activer le mode guillemets magiques (`--enable-magic-quotes`).
  - La fonction `get_magic_quotes_gpc()` retourne la configuration actuelle de l'option **`magic_quotes_gpc`** (0 pour l'option désactivée, 1 pour l'option activée) :

```
if(!get_magic_quotes_gpc()){
    $name = addslashes($_POST['name']);
} else {
    $name = $_POST['name'];
}
```



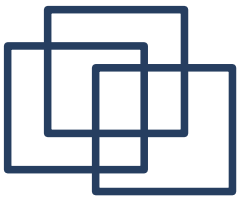
16 . Nettoyer du code HTML/PHP indésirable (2)

- **`htmlspecialchars()`** : convertit tous les caractères spéciaux en entité HTML.
- **`htmlentities()`** : convertit tous les caractères spéciaux en entité HTML.

```
print(htmlspecialchars($variable));
//ou
echo htmlspecialchars($variable);
```

- *Remarque:* on peut aussi utiliser les expressions régulières

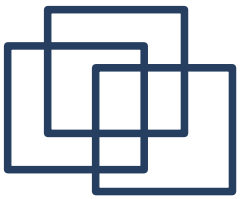
```
$html = preg_replace('/</', '&lt;', $html);
$html = preg_replace('/>/', '&gt;', $html);
```



16 . Nettoyer du code HTML/PHP indésirable (3)

- **`strip_tags()`** : enlève les balises HTML et PHP.

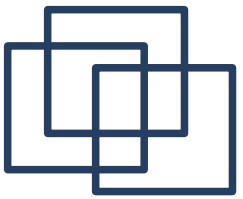
```
//balises autorisées :
$tags="<a><b><br><hr><i><img><p><table><td><th><tr>";
$html = strip_tags($html, $tags);
```



16 . Nettoyer du code HTML/PHP indésirable (4)

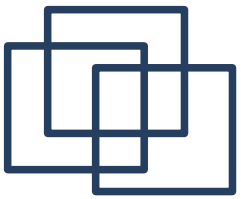
- Les expressions régulières :
  - PHP supporte deux types d'expressions régulières : POSIX et Perl. Le style Perl est lié à la librairie PCRE (*Perl Compatible Regular Expression*). Par défaut, PHP utilise la notation POSIX.
  - En savoir plus : http://no.php.net/manual/fr/ref.pcre.php
  - Les expressions régulières sont de très puissants outils de manipulation de textes et de données. Elles sont disponibles dans la plupart des langages (Perl, PHP, C, C++, Java, ...). Elles sont utilisées pour parcourir de façon automatique des textes à la recherche de morceaux de texte ayant certaines formes (un des exemples les plus répandus est celui de l'adresse email), et éventuellement remplacer ces morceaux de texte par d'autres :
  - *Exemple:* supprimer le CSS javascript

```
$html = preg_replace('`<script.*?/script>`', '', $html);
```



17 . Les template (1)

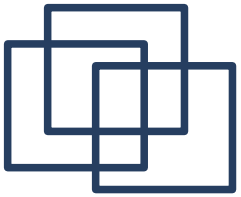
- Ils permettent de séparer la présentation (HTML) du contenu (PHP).
- Il faut éviter au maximum de mélanger les couches (architecture n-tiers). La meilleure utilisation est dans une situation où le(s) codeur(s) et le(s) designer(s) ne sont pas la même personne.
- Avec les templates, les designers ont 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).



17 .Les template (2)

- L'utilisation des template permet de capitaliser très rapidement du code recyclable car débarassé des contraintes de présentations, et d'offrire un accès direct au code HTML qui est facilement modifiable.
- Ce type de solution a fait ses preuves en PHP à travers des moteurs du type de **Smarty**, PHPlib, FastTemplates ou **ModeliXe** (http://modelixe.phpedit.com).
- Le framework **PEAR** (*PHP Extension and Application Repository*) propose aussi plusieurs solutions de templates :
  - HTML_Template_Flexy : *An extremely powerful Tokenizer driven Template engine*
  - HTML_Template_IT : *Integrated Templates*
  - HTML_Template_PHPLIB : *preg_* based template system.*
  - HTML_Template_Sigma : *An implementation of Integrated Templates API with template 'compilation' added*
  - HTML_Template_Xipe : *A simple, fast and powerful template engine.*

    (http://pear.php.net/packages.php?catpid=10&catname=HTML&pageID=2)



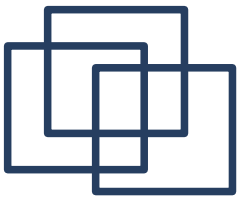
17 .Les template (3) : Smarty

- smarty est un moteur de template de la catégorie "PHP Code Builder" (http://smarty.php.net/)
- Le fichier index.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>
```



17 .Les template (4) : Flexy

- HTML_Template_Flexy est un moteur de template de la catégorie catégorie "Replacement systems" (http://pear.php.net/package/HTML_Template_Flexy)
- Le fichier index.php :

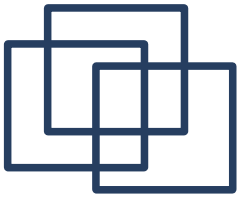
```
<?php
class example
{  var $tag = "hello world !"; }

$data = new example;
$output = new HTML_Template_Flexy();
$output->compile("hello.html");
$output->outputObject($data);
?>
```

- Le fichier hello.html :

```
<B>{tag}</B>      <B>{tag:h}</B>      <B>{tag:u}</B>
```

- Pour les variables, il existe trois utilisations :
  - # sans -> htmlspecialchars($tag)
  - # :h -> $tag tel quel (mode raw)
  - # :u -> urlencode($tag)



18 . Convention de codage

- Il existe des conventions de codage pour le PHP :
  - Une convention en anglais :
    http://alltasks.net/code/php_coding_standard.html
  - Une version moins détaillée sur : pear.php.net.
  - Une version allégée en fr : sur le serveur de la section

La fournir ou donner la référence de la convention appliquée