INTRODUCTION A PHP

1 INTRODUCTION

PHP, est un acronyme récursif : "PHP: Hypertext Preprocessor" ou "PHP Hypertext Preprocessing"
Un **langage de script, exécuté coté serveur**.
La syntaxe est emprunté aux langages C (principalement), Java et Perl
Projet open-source (le code source est accessible à tous et chacun peut y contribuer)
Les capacités de PHP vont au-delà de la génération dynamique de pages HTML / XHTML : PHP génère des documents PDF, des images GIF, JPEG ou PNG, des animations Flash à la volée, par exemple.

PHP comporte :

- **Une bibliothèque standard**
- **Des bibliothèques supplémentaires** tierces ou personnelles (des librairies qui peuvent être gratuites ou payantes)

1.1 Fonctionnalités (exemples)

- Fonctions mathématiques classiques
- Expressions régulières
- Gestion des fichiers et des répertoires
- Gestion du temps, de calendriers
- Fonctionnalités web et réseaux
- Gestion des formulaires web, de l'upload
- Gestion de protocoles Internet (HTTP, FTP, SMTP, par exemple)
- Analyse des documents XML
- Gestion des cookies
- Gestion des sessions
- Gestion des sockets
- Manipulation évoluée des chaînes et des fichiers HTML / XHTML
- Authentification http
- Manipulation des en-têtes http
- Accès natif a un grand nombre de bases de données (Informix, InterBase, mSQL, MySQL, Oracle, PostgreSQL, Sybase ...), accès en ODBC ou Adabas D aux bases de données non supportées en natif (DB/2 ou Access par exemple), lecture des fichiers DBase et FilePro
- Exécution de programmes externes et de commandes shell
- Encryptage
- Compression (gzip)
- Création de documents PDF (pour Adobe Acrobat Reader)
- Création et manipulation d'images GIF, JPEG et PNG
- Etc.

1.2 Origine du langage

Né avec le site de Rasmus Lerdof : permettait à l'origine de conserver une trace des visiteurs (1994).
Première version publique au début de l'année 1995.
"Personal Sommaire Page Tools" (d'autres indiquent "Personal Home Page"). Quelques utilitaires couramment utilisés dans les pages Web : livre d'or, compteur, etc.
Réécrit puis appelé PHP/FI Version 2. FI (Form Interpreter).
Fin 1996 : 15 000 sites Web
Courant 1997 : 50 000 + Devient projet d'équipe + Nouvel analyseur : base de la version 3.
En 2001 : prêt de 7 millions de sites.

PHP utilise le moteur d'analyse **Zend** (Conçu par Andi Gutmans & Zeev Suraski - ZEND - http://www.zend.com)

- Augmentation des performances,
- Support d'un nombre plus grand de librairies et extensions.
- Fonctionne sur les principaux serveurs Web.

Zend Engine : « compile puis exécute » (alors que PHP à la base : « exécute pendant le parsage »).

Principaux sites d'informations sur php
http://www.php.net
http://www.zend.com

NB : dans ce cours, nous nous intéressons à PHP 4 et PHP 5.

2 Intégration du code PHP

Un programme PHP s'exécute à partir d'un document d'instructions au format texte (ASCII), portant l'extension voulue (.php, .php5, .php4, .php3 ...) selon la configuration du serveur. L'extension .php est généralement la plus utilisée.

2.1 Intégration dans le document

Le code PHP est **intégré directement à l'intérieur d'un document HTML / XHTML ou utilisé indépendamment**

Le code PHP doit être inclus entre des balises :

- ```
  <?php
  print("La solution la plus courante.");
  ```
  ```

?>

- ```
  <script language="php">
  print ("Attention, certains éditeurs gèrent mal cette solution");
  </script>
  ```
- ```
  <?
  print("La solution de type SGML.");
  ?>
  ```
  NB : PHP doit être configuré ou compilé de manière particulière pour utiliser cette solution
- ```
  <%
  print ("La solution style ASP");
  %>
  ```
  NB : PHP doit être configuré ou compilé de manière particulière pour utiliser cette solution

En résumé : sauf pour une raison particulière, on privilégie la première solution.

Exemple
```
...
<body>
<?php
      print (" Hello World ");
?>
</body>
...
```

Le client ne reçoit que le résultat du script, non le code qui a produit ce résultat (différent de javaScript qui est exécuté coté client), à la condition, bien sûr, que le serveur soit configuré pour interpréter le PHP.
Si l'on affiche le code source reçu dans un navigateur, cela donnera :
```
...
<body>
Hello World
</body>
...
```

Remarque :
Il est possible de mêler des conditions PHP et du HTML / XHTML, par exemple :
```
...
<body>
<?php
$condition = false;
if ( $condition )
{
?>
Ceci est vrai
<?php
}
else
{
?>
Ceci est faux
<?php
}
?>
</body>
...
```

Affiche : Ceci est faux (code <p>Ceci est faux</p>).

PHP traite le texte à l'extérieur des balises PHP comme une fonction echo() (ou print), sans remplacer les variables éventuelles par leur valeur.

2.2 Séparateur d'instruction

Les instructions sont séparées par un point-virgule.

2.3 Commentaires

Ils sont

- encadrés par des /* et des */ lorsqu'ils tiennent sur plusieurs lignes
- précédés de // lorsqu'ils apparaissent sur une seule ligne

Exemple
```
<?php
/*
Mes commentaires tiennent sur plusieurs
plusieurs
lignes ...
```
```

```
*/
...
...
// Un commentaire sur une ligne
...
...
?>
```

3 TYPES DE DONNEES

PHP supporte **8 types basiques de données** :

Types scalaires :
- booléen
- entier
- nombre à virgule flottante (float, double)
- chaîne de caractères

Types composés :
- tableau
- objet

Types spéciaux :
- ressource
- null

3.1 Booléens

Valeurs true ou false

```
<?php
$monBooleen = true;
?>
```

Lors des conversions de types (voir ci-après), les valeurs suivantes sont considérées comme fausses (false) :
- false
- entier 0
- nombre à virgule flottante 0.0
- chaîne vide, et la chaîne "0"
- tableau vide (aucun élément)
- objet vide (aucun élément)
- constante null

3.2 Entiers

Il est possible de spécifier des nombres entiers en base :
- décimale (base 10)
- hexadécimale (base 16) - préfixés avec 0x
- octale (base 8) - préfixés avec un 0

Lesentiers négatifs sont déclarés préfixés avec le signe moins (-).

```
<?php
$valeur = 1;
$valeur = -2;
$valeur = 012;
$valeur = 0x12;
?>
```

Taille des entiers : selon la plate-forme
Pour mémoire, valeur maximale ~ un peu plus de 2 milliards
En cas de dépassement de capacité, le nombre sera interprété comme un double.

Lors des conversions de types (voir ci-après) :
- false devient 0
- true devient 1
- lors de conversion entre nombre à virgule flottante et un entier, le nombre sera arrondi à la valeur inférieure s'il est positif, et supérieure s'il est négatif (conversion 'vers zéro').

3.3 Nombres à virgule flottante (double ou float ou nombres réels)

```
<?php
$valeur = 1.1;
$valeur = 1.2;
```

```
?>
```

3.4 Les chaînes de caractères

Une chaîne peut être spécifiée de trois manières différentes :

guillemets simples

```
<?php
$valeur = 'texte';
$valeur = 'aujourd\'hui';
$valeur = 'Il a dit "bonjour" à toute la classe';
?>
```

NB : dans ce cas, les variables incluses dans la chaîne ne sont pas remplacées par leur valeur

```
<?php
$v= 'toto';
$valeur = 'La variable $v ne sera pas remplacée par sa valeur';
?>
```

guillemets doubles

```
<?php
$valeur = "texte";
$valeur = "Il a dit \"bonjour\" à toute la classe";
$valeur = "Il a dit 'bonjour' à toute la classe";
?>
```

NB : dans ce cas, les variables incluses dans la chaîne sont remplacées par leur valeur

```
<?php
$v = 'toto';
$valeur = "La variable $v est remplacée par sa valeur (c'est à dire toto)";
?>
```

syntaxe Here Doc ("documentation ici")

Une séquence :
<<<, suivi d'un identifiant arbitraire, puis d'une chaîne. Cette séquence se termine par l'identifiant initial, placé en premier sur une nouvelle ligne.

```
<?php
// Exemple 1
$chaine = <<<EOD
Exemple de chaîne
s'étalant sur
plusieurs lignes
avec la syntaxe heredoc
EOD;
print ($chaine);

// Exemple 2
print <<<EOD
Exemple de chaîne
s'étalant sur
plusieurs lignes
avec la syntaxe heredoc
EOD;

// Exemple 3
$v = 'Toto';
echo <<<EOT
Mon nom est "$v".
EOT;
?>
```

Remarque : syntaxe Here doc -> même comportement qu'une chaîne à guillemets doubles. Mais avec Here doc, inutile d'échapper les guillemets (simples ou doubles).

Lors d'une conversion de type

Evaluation comme une valeur numérique
La chaîne de caractères est de type "double" si elle contient un des caractère '.', 'e' ou 'E'.
Sinon, elle est de type entier ("integer").
La valeur est définie par la première partie de la chaîne. Si la chaîne de caractères débute par une valeur numérique cette valeur sera celle utilisée. Sinon, la valeur sera égale à 0 (zéro).

Lorsque la première expression est une chaîne de caractères, le type de la variable dépend de la seconde expression.

Exemples :

```
<?php
$valeur = 1 + "10.5"; // $valeur est du type double (11.5)
$valeur = 1 + "toto"; // $valeur est du type entier (1)
$valeur = 1 + "10 toto"; // $valeur est du type entier (11)
```

```
$valeur = "10.0 toto" + 1; // $valeur est du type integer (11)
$valeur = "10.0 toto" + 1.0; // $valeur est du type double (11)
?>
```

3.5 Les tableaux

Pour créer un tableau : **array()**
Cette fonction prend généralement en argument des structures **clé => valeur**, séparées par des virgules.

La clé est soit un entier (tableau indexé), soit une chaîne de caractères (tableau associatif).
La valeur contient n'importe quel type de données

array ([key1 =>] value1, [key2 =>] value2, ...)

Remarques :

- Si l'on n'indique pas de clé, l'indice maximum + 1 est utilisé comme clé par défaut (S'il n'y a pas encore eu d'indice numérique, l'indice est 0).
- Si l'on indique une clé déjà assignée, la nouvelle valeur écrase la précédente.

Exemples

```
<?php
$tableau = array('couleur' => 'rouge',
'goût' => 'sucre',
'forme' => 'rond',
'nom' => 'pomme',
4 // pas de clé -> la clé sera 0
);

// Est équivalent à
$tableau ['couleur'] = 'rouge';
$tableau ['goût'] = 'sucre';
$tableau ['forme'] = 'rond';
$tableau ['nom'] = 'pomme';
$tableau [] = 4; // pas de clé -> la clé sera 0
?>

<?php
$tableau[] = 'a';
$tableau[] = 'b';
$tableau[] = 'c';
// ou $tableau = array ('a' , 'b' , 'c')
// Crée 1 tableau (0 => 'a' , 1 => 'b' , 2 => 'c')
?>

<?php
$tableau = array (2, 4, 6, 8, 10);
// ceci est la même chose que array( 0 => 2, 1 => 4, ...)
?>

<?php
$tableau = array (10,         // clé = 0
5 => 6,
3 => 7,
'a' => 4,
11,                   // clé = 6 (index maximum : 5)
'8' => 2,             // clé = 8 (entier)
'02' => 77,           // clé = '02'
0 => 12        // la valeur de la clé 0 sera remplacée par 12
);
?>

<?php
// Tableau vide
$empty = array();
?>
```

Pour modifier un tableau existant : on lui affecte des valeurs.
L'affectation de valeurs de tableau se fait en spécifiant la clé entre crochets.
Si l'on n'indique pas de clé ($tableau[]), la valeur est ajoutée à la fin du tableau.
$tableau[key] = value;
$tableau[] = value;

NB : Si $tableau n'existe pas, il sera créé.

foreach

foreach est un type de boucle dédié aux tableau : passer "en revue" les valeurs d'un tableau.

```
<?php
$tableau = array ('rouge','bleu','vert','jaune');
foreach ( $tableau as $couleur )
{
   echo "La couleur est $couleur\n";
}
?>
```

```
Affiche :
La couleur est rouge
La couleur est bleu
La couleur est vert
La couleur est jaune
```

Remplir un tableau

```
<?php
// remplis un tableau avec les entiers de 10 à 49
$tableau=array();
for ($i=10; $i<50; ++$i)
{
   $tableau[] = $i;
}
?>
```

Les tableaux sont ordonnés.
Il existe plusieurs fonctions de classement.
Exemple

```
<?php
sort($tableau);
?>
```

3.6 Tableaux multidimensionnels, et récursifs

```
<?php
// php est extrêmement souple, l'exemple ci-dessous n'est généralement pas à suivre !
$tableau = array ("fruits" => array ("a" => "orange",
                                     "b" => "banane",
                                     "c" => "pomme"),
                                     "nombres" => array (1, 2, 3, 4),
                                     "couleur" => array ("jaune",
                                                         5 => "vert",
                                                         "bleu"));
?>
```

3.7 Les objets

Se définissent comme une classe (nous y revenons plus loin)

Initialisation d'un objet : la commande **new** crée l'instance de l'objet.

```
<?php
class Demo
{
   function afficher ()
   {
      echo "Hello";
   }
}
$test = new Demo();
$test->afficher();
?>
```

3.8 Ressources

type spécial = référence sur une ressource externe.
Les ressources sont créées par des fonctions dédiées.

Exemple
ftp_connect : ouvre une connexion FTP
resource ftp_connect (string host, int [port])

ftp_connect() retourne un flot FTP en cas de succès, et FALSE sinon.
ftp_connect() ouvre une connexion FTP avec l'hôte host. Le paramètre port spécifie le port de connexion. S'il est omis, le port 21 sera utilisé

3.9 NULL

NULL = **absence de valeur**

Constante, insensible à la casse.

```
<?php
$valeur = NULL;
?>
```

Remarque : la fonction unset permet de détruire une variable

3.10 Définition du type

PHP ne nécessite **pas de déclaration explicite** du type d'une variable.
Le type d'une variable est déterminé par le contexte d'utilisation.

Exemples avec l'opérateur '+' :

```
<?php
$valeur = "0"; // $valeur est une chaîne de caractères
$valeur += 2; // $valeur est du type entier (2)
$valeur = $valeur + 1.3; // $valeur est du type double (3.3)
$valeur = 5 + "10 toto"; // $valeur est du type entier (15)
?>
```

Fonction gettype
string gettype (mixed var)
gettype($v) retourne le type de la variable.
Remarque mixed est une convention de notation qui signifie que le type de la variable peut être divers

La fonction retourne une chaîne de caractères indiquant le type :

- "boolean"
- "integer"
- "double"
- "string"
- "array"
- "object"
- "resource"
- "user function"
- "unknown type"

Fonction settype
int settype (string var, string type)
settype affecte un type à une variable.

Les valeurs possibles pour le paramètre type sont :

- "integer"
- "double"
- "string"
- "array"
- "object"

La fonction renvoie TRUE en cas de succès, FALSE sinon.

Transtypage
La conversion de type en PHP fonctionne de la même manière qu'en C : le nom du type voulu est écrit entre parenthèses devant la variable à transtyper ("cast").

```
<?php
$valeur = 10; // $valeur est un entier
$autreValeur = (double) $valeur ; // $autreValeur est un double
?>
```

Les conversions autorisées sont:

- (int), (integer) - type entier
- (bool), (boolean) – booléen
- (real), (double), (float) - type double
- (string) - ctype chaîne
- (array) - type tableau
- (object) - type objet

Remarques :
Lors du transtypage d'un scalaire ou une chaîne en tableau, la valeur de la variable est affectée au premier élément du tableau.

```
<?php
$valeur = 'toto';
$valeur = (array) $valeur;
echo $valeur [0];
// affiche 'toto'
?>
```

Lors du transtypage d'un scalaire ou une chaîne en objet, la valeur de la variable est transformée en attribut de l'objet. L'attribut s'appelle 'scalar':

```
<?php
$valeur = 'toto';
$objet = (object) $valeur;
echo $objet ->scalar;
// affiche 'toto'
?>
```

NB : le transtypage n'a pas toujours de résultat prévisible. Etre prudent.

Il existe les fonctions strval, doubleval, intval pour obtenir respectivement la valeur d'une variable sous forme de chaîne, de double ou d'entier.

4 VARIABLES

4.1 Généralités

Un signe dollar "$" suivi du nom de la variable.

Attention : le nom est sensible à la casse ($toto != $Toto).
Un nom de variable valide doit commencer par une lettre ou un souligné (_), suivi de lettres, chiffres ou soulignés.

Les variables sont **assignées par valeur** (lorsque l'on assigne une expression à une variable, la valeur de l'expression est recopiée dans la variable).

PHP permet d'assigner les valeurs aux variables **par référence**.
La nouvelle variable référence (en d'autres termes, "devient un alias de", ou encore "pointe sur") la variable originale.
Les modifications d'une des variables affecteront l'autre.
Aucune copie n'est faite (assignation plus rapide, notamment pour de grands objets, comme les tableaux).

4.2 Assignation par référence

Avec un & (ET commercial) au début de la variable

```
<?php
$v1 = 'Toto';
$v2 = &$v1;
$v2 = "Titi";
echo $v1 ;
echo "<br />";
echo $v2;
?>
```

4.3 Portée des variables

Portée **globale** : variables définies en dehors des fonctions

Portée **locale** : variables définies à l'intérieur d'une fonction (la portée de la variable est limitée à cette fonction)

```
<?php
$v = 1;
//... suite du script
?>
```

La variable $valeur est accessible dans la suite du script, mais :

```
<?php
$v = 1;
function test()
{
   echo $v;
}
test();
?>
```

Le script n'affiche rien à l'écran
Explication : echo() utilise la variable locale $v, celle-ci n'a pas été assignée préalablement dans la fonction.

Différent d'autres langages (une variable globale est automatiquement accessible dans les fonctions, à moins d'être redéfinie localement dans la fonction).

En PHP, une variable globale doit être déclarée globale à l'intérieur de la fonction afin de pouvoir être utilisée dans cette fonction.

```
<?php
$v1 = 1;
$v2 = 2;
```

```
function somme()
{
   global $v1;
   global $v2;
   $somme = $v1  + $v;
}
somme();
echo $somme ;
?>
```

Les variables $ v1et $ v1sont déclarées globales dans la fonction somme() : toutes les références à ces variables concerneront les variables globales.

4.3.1 Variables "static"

Une variable statique a une portée locale uniquement, mais elle **garde sa valeur au fil des appels de la fonction**.
Déclarée avec le mot clé static

Exemple sans static

```
<?php
function test()
{
  $v = 0;
  echo $v;
  echo "<br />";
  $v++;
}
?>
```

A chaque appel $v est affecté à 0 et la fonction affiche "0". Dès que la fonction est terminée la variable disparaît.

Exemple avec static

```
<?php
function test()
{
   static $v = 0;
   echo $v ;
   echo "<br />";
   $v ++;
}
test();
test();
test();
?>
```

```
Affiche
0
1
2
```

Très utile lors des appels récursifs de fonctions

```
<?php
function test()
{
   static $count = 0;
   $count++;
   echo $count;
   echo "<br />";
   if ($count < 10)
   {
     test();
   }
   $count--;
   echo $count;
   echo "<br />";
}
?>
```

4.3.2 Les variables dynamiques

~ **Noms de variables variables.**
Le nom de variable est affecté et utilisé dynamiquement.

Une variable dynamique prend la valeur d'une variable et l'utilise comme nom d'une autre variable.
Dans l'exemple ci-dessous, Hello peut être utilisé comme le nom d'une variable en utilisant le "$$" précédent la variable.

```
<?php
```

```
$valeur = "Hello";
$$valeur = "World";
?>
```

Deux variables ont été définies :
$valeur (valeur = "Hello")
$Hello (valeur = "World")

4.3.3 Variables externes à PHP & Formulaires HTML / XHTML (GET et POST) & URL

Lorsqu'un formulaire est envoyé à un script PHP, toutes les variables du formulaire seront automatiquement disponibles dans le script.

```
<form action="test.php" method="POST">
<input type="text" name="nom" id="nom /><br />
<input type="submit" />
</form>
```

Les variables peuvent également être transmises par l'appel d'un URL, par un lien

```
<a href="script.php?nom=valeur">...</a>
```

Pour accéder à la valeur de « nom » en PHP

Depuis PHP 4.1.0

$_POST['nom'] pour un formulaire en méthode POST
ou **$_GET['nom']** dans le cas d'un formulaire en méthode GET ou d'un lien

$_REQUEST['nom']
(Un tableau associatif constitué du contenu des variables $_GET, $_POST, $_COOKIE, et $_FILES)

Il existe aussi la méthode import_request_variables (voir documentation en ligne)

Avant PHP 4.1.0

$HTTP_POST_VARS['nom'] pour un formulaire en méthode POST
ou $HTTP_GET_VARS['nom'] dans le cas d'un formulaire en méthode GET ou d'un lien

Si la directive (configuration PHP) : register_globals vaut « on », alors :

$nom

Attention register_globals vaut « off » par défaut depuis PHP 4.2.0 (et il est conseillé de laisser cette valeur à off)

NB : Il est possible de passer des **tableaux de valeurs** depuis un formulaire

```
<form action="test.php" method="POST">
<select multiple name="cours[]" id="cours" >
   <option value="programmation">Prog</option>
   <option value="algorithmes">Algo</option>
   <option value="sql">SQL</option>
</select><br />
<input type="submit" />
</form>
```

$_POST['cours'] contiendra un tableau contenant les différentes valeurs sélectionnées.

4.3.4 Cookies

Mécanisme permettant de stocker des données sur la machine cliente (souvent pour identifier un utilisateur - en fait un utilisateur utilisant un navigateur sur un ordinateur).

setcookie(params) permet d'envoyer un cookie.

```
<?php
setcookie("TestCookie","Toto");
?>
```

Tout cookie envoyé depuis le client sur le serveur est stocké sous forme de variable, comme pour la méthode POST ou GET.

Pour afficher le cookie (dans une autre page)

```
<?php
echo $_COOKIE["TestCookie"];
?>
```

Un cookie remplace le cookie précédent par un cookie de même nom tant que le "path" ou le domaine sont identiques (voir prototype de la méthode setcookie).

4.3.5 Variables prédéfinies

Dépendent
- du serveur
- de la version du serveur
- de la configuration du serveur
- d'autres facteurs

Certaines de ces variables ne sont pas accessibles lorsque PHP fonctionne en exécutable.
Pour obtenir une liste des variables prédéfinies dans le contexte : **fonction phpinfo()**.

Voir la documentation PHP.

5 Constantes

Identifiant qui représente une valeur simple.
Seuls les types de **données scalaires** peuvent être placés dans une constante : c'est à dire les types booléen, entier, double et chaîne de caractères
Leur valeur ne peut **jamais être modifiée** durant l'exécution du script

NB :
- le nom d'une constante est **sensible à la casse**
- par **convention**, les noms de constantes sont en **majuscules**

Un nom de constante valide commence par une lettre ou un souligné (_), suivi d'un nombre quelconque de lettre, chiffres ou soulignés.

Les constantes ont une **portée globale**.

Syntaxe
```
<?php
define("CONSTANTE", "Hello World");
echo CONSTANTE;
?>
```

```
Affiche
Hello World
```

NB : on ne préfixe pas le nom de la constante avec $.

6 Opérateurs

Les opérateurs arithmétiques

Opérateur	Résultat
+	ADDITION D'OPERANDES
-	Soustraction d'opérandes
*	Multiplication d'opérandes
/	Division d'un opérande par un autre opérande
%	Reste d'une division
--	Augmente la valeur d'un opérande d'une unité
++	Diminue la valeur d'un opérande d'une unité

Les opérateurs d'assignement

Opérateur	Description
=	ASSIGNE LA VALEUR DE L'OPERANDE DROIT A L'OPERANDE DE GAUCHE
+=	Additionne la valeur de l'opérande droit à la valeur de l'opérande gauche et assigne le résultat à l'opérande de gauche
-=	Soustrait la valeur de l'opérande droit de la valeur de l'opérande gauche et assigne le résultat à l'opérande de gauche
*=	Multiplie la valeur de l'opérande droit et la valeur de l'opérande gauche et assigne le résultat à l'opérande de gauche
/=	Divise la valeur de l'opérande gauche par la valeur de l'opérande droit et assigne le résultat à l'opérande de gauche
%=	Divise la valeur de l'opérande gauche par la valeur de l'opérande droit et assigne le reste à l'opérande de gauche
&=	La valeur de l'opérande gauche devient la valeur de l'opérande gauche bitwise AND la valeur de l'opérande droite
|=	La valeur de l'opérande gauche devient la valeur de l'opérande gauche bitwise OR la valeur de l'opérande

	droite
^=	La valeur de l'opérande gauche devient la valeur de l'opérande gauche bitwise XOR la valeur de l'opérande droite
.=	La valeur de l'opérande gauche devient la valeur de l'opérande gauche concatenée à la valeur de l'opérande droite

Les opérateurs sur les bits (opérateurs bitwise)
Les opérateurs sur les bits permettent de manipuler les bits dans un entier.

Opérateur	Description
&	ET (AND) Les bits positionnés à 1 dans l'opérande gauche ET dans l'opérande droite sont positionnés à 1.
|	OU (OR) Les bits positionnés à 1 dans l'opérande gauche OU l'opérande droite sont sont positionnés à 1
^	Xor Les bits positionnés à 1 dans l'opérande gauche OU dans l'opérande droite sont positionnés à 1.
~	NON (Not) Les bits qui sont positionnés à 1 dans l'opérande sont positionnés à 0, et vice versa.
<<	Décalage à gauche Décale les bits de l'opérande gauche dans l'opérande droite par la gauche (chaque décalage équivaut à une multiplication par 2).
>>	Décalage à droite Décalage des bits de l'opérande gauche dans l'opérande droite par la droite (chaque décalage équivaut à une division par 2).

Exemples
L'opérateur & entre 1 et 2

	0	**0**	**0**	**1**	**(1)**
&	**0**	**0**	**1**	**0**	**(2)**
	0	**0**	**0**	**0**	**(0)**

L'opérateur & entre 1 et 3

	0	**0**	**0**	**1**	**(1)**
&	**0**	**0**	**1**	**1**	**(3)**
	0	**0**	**0**	**1**	**(1)**

L'opérateur & entre 1 et 4

	0	**0**	**0**	**1**	**(1)**
&	**0**	**1**	**0**	**0**	**(4)**
	0	**0**	**0**	**0**	**(0)**

L'opérateur & entre 1 et 5

	0	**0**	**0**	**1**	**(1)**
&	**0**	**1**	**0**	**1**	**(5)**
	0	**0**	**0**	**1**	**(1)**

Les opérateurs de comparaison

Opérateur	Description
==	Retourne true si les opérandes sont égaux (valeur)
===	Retourne true si les opérandes sont égaux (valeur ET type de données)
!= **<>**	Retourne true si les opérandes ne sont pas égaux
>	Retourne true si l'opérande gauche est supérieur à l'opérande droit
<	Retourne true si l'opérande gauche est inférieur à l'opérande droit
>=	Retourne true si l'opérande gauche est supérieur ou égal à l'opérande droit
<=	Retourne true si l'opérande gauche est inférieur ou égal à l'opérande droit
?	Expression conditionnelle ternaire ($a==$b)?valeurSiVrai:valeurSiFaux

L'opérateur de contrôle d'erreur

Opérateur	Description
@	Suppression des messages d'erreur lorsqu'il est ajouté avant une expression (variables, fonctions, include(), constantes, ...)

L'opérateur d'exécution

Opérateur	Description
``	(Guillemets obliques) PHP exécuter le contenu de ces guillemets obliques comme une commande shell. Retourne le résultat

```
<?php
$liste = `ls`;
echo "<pre>$liste</pre>";
?>
```

NB : Cet opérateur est désactivé lorsque le "safe mode" de PHP est activé (configuration de PHP).

Les opérateurs logiques

Opérateur	Description
and &&	Retourne true si l'opérande gauche ET l'opérande droit retourne la valeur true (and et && ont des priorités différentes)
or ||	Retourne true si au moins un des opérantes retourne true (or et || ont des priorités différentes)
xor	Retourne true si au un des opérantes retourne true, mais pas les deux (ou exclusif)
!	Retourne true si l'expression a la valeur false

Les opérateurs de chaînes

Opérateur	Description
.	Concaténation de chaînes de caractères
.=	Concaténation de chaîne de caractères (ajoutées à l'opérande gauche)

Opérateurs divers

Opérateur	Description
$	Référence une variable
&	Référence l'adresse d'une variable
->	Référence une méthode ou propriété de classe
=>	Assigne un index à un élément de tableau

Priorité des principaux opérateurs

	Opérateur
Du + élevé	! ~ ++ -- $
	* / %
	+ -
	< <= > >=
	=== == !=
	&
	^
	|
	&&
	||
	= += -= *= /= .= %= &= |= ^=
	AND
	XOR
Au moins élevé	OR

7 Structures de contrôle

7.1 if

```
if (expression conditionnelle)
{
   commandes
}
```

7.2 if ... else

```
if (expression conditionnelle)
{
   commandes;
}
else
{
   commandes;
}
```

7.3 if ... elseif

Combinaison de if et de else.

```
if (expression conditionnelle)
{
   commandes;
}
elseif (expression conditionnelle)
{
   commandes;
}
elseif (expression conditionnelle)
{
   commandes;
}
```

7.4 switch

```
switch (expression)
{
   case etiquette1 :
      commandes;
      break;
   case etiquette2 :
      commandes;
      break;
   default :
      commandes;
      break;
}
```

7.5 while

```
while (expression conditionnelle)
{
   commandes;
}
```

7.6 do ... while

```
do
{
   commandes;
} while (expression conditionnelle)
```

7.7 for

```
for (expression d'initialisation; expression conditionnelle; mise à jour)
{
   commandes;
}
```

7.8 foreach

```
foreach($tableau as $valeur)
{
   commandes;
}
foreach($tableau as $cle => $valeur)
```

```
{
   commandes;
}
```

7.9 break

Permet de quitter une structure for, while, foreach ou switch (sous réserve qu'une condition soit remplie par exemple)

```
while (expression conditionnelle)
{
   commandes;
   if (condition)
  {
      break;
  }
}
```

break accepte un argument numérique optionnel qui indique combien de structures emboîtées sont interrompues.

```
while (expression conditionnelle)
{
   while (expression conditionnelle)
   {
      commandes;
      if (condition)
     {
         break 2;
     }
   }
}
```

7.10 continue

Permet de passer directement à l'itération suivante de la boucle

```
while (expression conditionnelle)
{
   commandes;
   if (condition)
   {
      continue;
      // si la condition vaut true,
      // les commandes ci dessous ne seront
      // pas exécutées pour cette boucle
   }
   commandes;
}
```

continue accepte un argument numérique optionnel qui indique combien de structures emboîtées sont interrompues.

```
while (expression conditionnelle)
{
   commandes;
   while (1)  // toujours vrai
   {
      break 2;
   }
   jamaisAtteint;
}
```

7.11 require()

require(fichier) est remplacée par le contenu du fichier spécifié
(cf. #include du C)

Le code qui est dans le fichier doit être placé entre les balises habituelles de PHP.

require est toujours exécutée (cela ne sert à rien de le placer dans une condition).
Toute variable qui est dans le champs du script sera accessible dans le fichier d'inclusion, et vice-versa.

7.12 include()

include(fichier) inclus et évalue le fichier spécifié en argument
Le code qui est dans le fichier doit être placé entre les balises habituelles de PHP.
include peut être placé dans une condition, elle ne sera exécutée que si la condition est remplie
Le fichier est inclus autant de fois que include(fichier) est rencontré

Toute variable qui est dans le champs du script sera accessible dans le fichier d'inclusion, et vice-versa.

7.13 require_once()

require_once(fichier) est remplacée par le contenu du fichier spécifié

Le code ne sera ajouté qu'une seule fois même si require_once est rencontré plusieurs fois pour le même fichier (contrairement à require) : évite les redéfinitions de variables ou de fonctions, génératrices d'alertes de PHP.

7.14 include_once()

include_once(fichier) inclus et évalue le fichier spécifié en argument
Le code ne sera inclus qu'une seule fois même si include_once est rencontré plusieurs fois pour le même fichier (contrairement à include) : évite les redéfinitions de variables ou de fonctions, génératrices d'alertes de PHP.

8 Fonctions

8.1.1 Généralités

Une fonction utilisateur est définie en utilisant la syntaxe suivante :
function nomFonction (paramOptionnels)
{
commandes;
retour optionnel;
}

Exemple
```
<?php
function demo ($param1, $param2, ...)
{
   $retVal = "valeur à retourner";
   return $retVal;
}
?>
```

8.1.2 Les arguments de fonctions

Chaque argument est séparé par une virgule.

PHP supporte le passage d'arguments
- par **valeur** (méthode par défaut)

par référence

Remarque : PHP 4 (et + récent) supporte les listes variables d'arguments (voir les fonctions func_num_args(), func_get_arg(), et func_get_args())
Exemple
```
<?php
function argumentaire()
{
    $numArgs = func_num_args();
    echo "Nombre d'arguments: $numArgs";
}
argumentaire (1, 2, 3);  // affiche 'Nombre d'arguments: 3'
?>
```

NB : on peut arriver au même résultat en passant un **tableau comme argument**

8.1.3 Passage d'arguments par référence

Par défaut, les arguments sont passés à la fonction par valeur (si la valeur d'un argument change dans la fonction, elle ne change pas à l'extérieur de la fonction).
Pour passer un argument par référence : ajouter un **& devant l'argument**

Exemple avec passage par valeur
```
<?php
function concatene($chaine)
{
   $chaine .= ' concatenée';
}
$uneChaine = 'Une chaîne';
concatene($uneChaine);
echo $uneChaine ;
?>
```

Exemple avec passage par référence
```
<?php
function concatene(&$chaine)
{
   $chaine .= ' concatenée';
}
$uneChaine = 'Une chaîne';
concatene($uneChaine);
echo $uneChaine ;
?>
```

Remarque :
Il est possible de passer **une variable par référence à une fonction de manière ponctuelle** : ajouter un **& devant l'argument** dans l'appel de la fonction

Exemple
```
<?php
function concatene ($chaine)
{
   $chaine .= ' concatenée';
}
$uneChaine = 'Une chaîne';
concatene ($uneChaine );
echo $uneChaine ; // affiche Une chaîne
concatene (&$uneChaine );
echo $uneChaine ; // affiche Une chaîne concatenée
?>
```

8.1.4 Valeur par défaut des arguments

Il est possible de définir des valeurs par défaut pour les arguments de type scalaire

Exemple
```
<?php
function direQuelqueChose ($message = "Bonjour")
{
  echo "$message";
}
echo direQuelqueChose (); // Affiche Bonjour
echo "<br />";
echo direQuelqueChose ("Au revoir"); // Affiche Au revoir
?>
```

La valeur par défaut d'un argument **doit obligatoirement être une constante**, et ne peut être ni une variable, ni un membre de classe.

Les arguments ayant une valeur par défaut doivent être **placés à la fin.**

```
<?php
function direQuelqueChoseAQuelqun ($message = "Bonjour", $aQui)
{
  echo "$message $aQui";
}
echo direQuelqueChoseAQuelqun ("Toto");
?>
```
Affiche un avertissement : il manque l'argument 2
Et affiche toto comme valeur de l'argument 1

```
<?php
function direQuelqueChoseAQuelqun ($aQui, $message = "Bonjour")
{
  echo "$message $aQui";
}
echo direQuelqueChoseAQuelqun ("Toto");
?>
```
Ca marche

8.1.5 Les valeurs de retour

Les valeurs sont renvoyées en utilisant une instruction de retour optionnelle (mot clé **return**).
Tous les types de variables peuvent être renvoyés, tableaux et objets compris

```
<?php
function carre ($num)
{
   return $num * $num;
}
echo carre (4); // affiche '16'.
?>
```

On ne peut pas renvoyer plusieurs valeurs en même temps (mais un tableau oui)

8.1.6 Fonction-variable

Si le nom d'une variable est suivi de parenthèses, PHP cherche une fonction de même nom, et essaie de l'exécuter

```
<?php
function vFonction()
```

```
{
    echo "Hello<br />";
}
function vAutreFonction ($arg)
{
   echo "$arg<br />";
}
$maFonction = 'vFonction';
$maFonction ();
$maFonction = 'vAutreFonction';
$maFonction ('test');
?>
```

9 Classes & objets

Une classe est déclarée en utilisant le mot clé **class**
Les classes forment **un type de variable**.

Un objet : instanciation d'une classe (objet ou instance de classe)
Un objet de la classe est déclarée en utilisant l'opérateur **new**

Une classe est composée de deux parties:

- Les **attributs** (ou **données membres**): données représentant l'état de l'objet
- Les **méthodes** (ou **fonctions membres**): opérations applicables aux objets

9.1 Classes et objets avec PHP 4

Remarque : en PHP 4, les attributs et méthodes sont toujours publics.

9.1.1 Déclaration d'une classe

```
class NomClasse
{
   // Données membres
   var $donnee1;
   var $donnee2;

   // Méthodes
   function fonction1(parametres)
   {
      commandes;
   }
}
```

Exemple

```
<?php
class Demo
{
   var $chaine = "Hello World";
   function initialiser($uneChaine)
   {
      $this->chaine = $uneChaine;
   }
   function afficher()
   {
      print ($this->chaine);
   }
}
$maDemo = new Demo;
print $maDemo->chaine;
$chaine->initialiser("Hello");
print $maDemo->chaine;
$maDemo->afficher();
?>
```

```
Affiche
Hello World
Hello
Hello
```

9.1.2 Attributs

Ils sont déclarés avec **var**
NB : seuls les initialiseurs constants sont autorisés pour les attributs.

Cet exemple n'est pas correct

```
<?php
```

```
class Demo
{
   // non correct
   var $demoVar = $autreVariable;
   // non correct
   var $demoChaine = $ch1 . $ch2;
   // non correct
   var demoTableau = array ("un", "deux", "trois");
   // non correct
   var $demoDate = date("d/m/Y");
   // correct
   var $demoInt = 1;
}
?>
```

On peut utiliser les constructeurs pour les initialisations variables.
Cet exemple est correct

```
<?php
class Demo
{
   var $demoVar;
   var $demoChaine;
   var demoTableau;
   var $demoDate;

   function Demo($autreVariable, $ch1, $ch2)
   {
      $this->demoVar= $autreVariable;
      $this->demoChaine = $ch1 . $ch2;
      $this->demoTableau = array ("un", "deux", "trois");
      $this->demoDate = date("d/m/Y");
   }
}
?>
```

9.1.3 Accéder aux attributs d'un objet

$nomObjet->nomAttribut (sans le signe $ devant l'attribut)

Accéder aux méthodes d'un objet
nomObjet->nomMethode

$this
Désigner l'objet dans lequel on se trouve

Une fonction membre peut faire référence aux propriétés situées dans le même l'objet.

$this ~ l'objet courant

9.1.4 Héritage

Une classe peut être une extension d'une autre classe.
Mot clé : **extends**

La classe dérivée

- hérite de toutes les méthodes et attributs de la classe de base
- peut définir ses propres fonctions et attributs

NB : l'héritage multiple n'est pas supporté en PHP 4

9.1.5 Constructeur

Constructeur : fonction qui est **appelée automatiquement par la classe lors de la création d'une nouvelle instance d'une classe (new)**.
La fonction constructeur a le **même nom que la classe en PHP 4**

Exemple

```
<?php
class Demo
{
   $var1;
   $var2;

   function Demo ($v1,$v2)
   {
      $this->var1=$v1;
      $this->var2=$v2;
```

```
    }
}
?>
```
Note : Il n'y a pas de destructeur en PHP 4

9.1.6 Opérateur :: (contexte de classe)

L'opérateur :: sert à

- Faire référence aux méthodes et attributs d'une classe de base (classe « parente », celle définie avec extends)
- Utiliser des méthodes de classes qui n'ont pas encore d'objets créés

Il s'utilise en préfixant l'appel de la méthode ou de l'attribut par le nom de la classe et de ::
Par exemple :
MaClasse::maMethode() ;

9.1.7 Opérateur parent

L'opérateur parent permet de faire référence à la classe « parente » (celle définie avec extends), sans la nommer explicitement. Cela apporte de la souplesse si le contexte de l 'héritage (ou nom de la classe « parente ») est amené à changer.

9.2 Classes et objets avec PHP 5

Avec la version 5 de PHP, le modèle objet a beaucoup évolué.

9.2.1 Constructeurs

Constructeur : fonction qui est **appelée automatiquement par la classe lors de la création d'une nouvelle instance d'une classe (new)**.
Contrairement à PHP 4, la fonction constructeur **n'a pas** le même nom que la classe, mais se déclare avec __**construct()**

Exemple

```
<?php
class Demo
{
   $var1;
   $var2;

   function __construct($v1, $v2)
   {
      $this->var1=$v1;
      $this->var2=$v2;
   }
}
?>
```

NB :

- Le constructeur parent n'est pas appelé implicitement. Pour invoquer explicitement un constructeur parent, il faut utiliser parent::__construct().
- Si PHP 5 ne peut pas trouver une fonction __construct() pour une classe, il cherchera une fonction constructeur représentée par le nom de la classe (compatibilité ascendante avec PHP 4).

9.2.2 Destructeurs

Destructeur : **méthode appelée lorsqu'un objet est détruit**.

```
<?php
class Demo
{
   $var1;
   $var2;

   function __construct($v1, $v2)
   {
      $this->var1=$v1;
      $this->var2=$v2;
   }

   function __destruct()
   {
       print "Destruction de l'objet";
   }
}
?>
```
NB :

- Le destructeur parent n'est pas appelé implicitement. Pour invoquer explicitement le destructeur parent, il faut utiliser parent::__ destruct().

9.2.3 Visibilité des membres

Les données membres ou les méthodes peuvent être définies en préfixant leur déclaration avec :

- **public** : l'accès à ces éléments est « public », donc accessible par n'importe quelle partie d'un programme.

- **protected** : l'accès à ces élément est « protégé », c'est à dire limité aux classes héritées et parentes (et la classe elle-même).
- **private** : l'accès aux éléments « privés » est limité à la classe qui les a définis.

9.2.4 Opérateur de résolution de portée ::

L'opérateur de résolution de portée permet d'accéder aux membres statiques ou constants et aux élément « surchargés » (redéfinis) par la classe.

9.2.5 Eléments static

Les éléments static d'une classe sont des éléments indépendants de l'instanciation d'un objet ou de son contexte (et il y possible d'y accéder sans instancier l'objet).

La déclaration static est faite après la déclaration de visibilité (public / protected / private).

```
<?php
class Demo
{
    public static $vstatic = "toto";

    public static function valeurStatique()
    {
        return "valeur";
    }
}
…
print Demo::$vstatic;
print Demo::valeurStatique();
?>
```

NB: On **ne peut pas** accéder à des propriétés statiques à travers l'objet en utilisant l'opérateur ->.

9.2.6 Constantes

Les constantes sont des valeurs constantes qui restent inchangées tout au long de la vie de l' objet.

```
<?php
class Demo
{
   $var1;
   const HW = 'Hello World';
   ...
}
…
print Demo::HW;
?>
```

NB: On **ne peut pas** accéder à une constante à travers l'objet en utilisant l'opérateur ->.

9.2.7 Classes abstraites

PHP 5 permet les classes et les méthodes abstraites.
Une classe comprenant au moins une méthode abstraite doit être abstraite.
Seul la signature d'une méthode abstraite doit être déclarée (aucune implémentation). Elle doit être redéfinie dans la classe enfant, avec une visibilité identique ou plus faible (par exemple, une méthode abstraite définie comme protégée doit être redéfinie en tant que protégée ou publique).

```
<?php
abstract class Demo
{
    abstract protected function f1();
    …
}

class Demo2 extends Demo
{
     public function f1()
     {
       code...;
     }

     public function f2($v)
     {
       code...;
     }
}
?>
```

9.2.8 Interfaces

Les interfaces permettent de spécifier les méthodes et variables qu'une classe peut implémenter, sans définir comment ces

méthodes seront gérées.
Elles sont définies avec le mot clé **interface**
NB : toutes les méthodes déclarées dans une interface doivent être publiques.

implements permet d'implémenter une interface préalablement définie. Toutes les méthodes de l'interface doivent être implémentées dans une classe.

```
<?php
interface iDemo
{
   public function f1($v1, $v2);
   public function f2($v);
}

class maDemo implements iDemo
{
   private $vars;

   public function f1($v1, $v2);
   {
       code...;
   }

   public function f2($v)
   {
       code...;
   }
}
```

9.2.9 "final"

Le mot-clé "final" empêche les classes filles de surcharger (ou redéfinir) une méthode dans les classes « enfants ».
Une classe définie comme finale ne peut pas être étendue.

```
<?php
class Demo
{
   final public function f1()
   {
     code...;
   }
}
?>
```