INTRODUCTION A PHP

- **Objet du cours**
  - **Bases du langage PHP**.
  - Pré requis souhaitables :
    - Des connaissances du HTML et/ou du XHTML
    - Une première pratique de la composition de documents Web.
    - Des connaissances sur le fonctionnement d'Internet et des serveurs HTTP

Introduction

- **PHP** = acronyme récursif :
  - PHP: Hypertext Preprocessor
  - PHP Hypertext Preprocessing
  - Un **langage de script, exécuté coté serveur**.
- Projet open-source

Introduction

- Les capacités de PHP vont au-delà de la génération dynamique de pages HTML / XHTML :
  - génère des documents PDF, des images GIF, JPEG ou PNG, des animations Flash
  - etc.

- PHP comporte :
  - **Une bibliothèque standard**
  - **Des bibliothèques supplémentaires** tierces ou personnelles

Introduction

- 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

Introduction

- Fonctionnalités (exemples)
  - 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

Introduction

- Fonctionnalités (exemples)
  - 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.

Introduction

- Origine du langage
  - Né avec le site de Rasmus Lerdof :
    - à l'origine : conserver une trace des visiteurs (1994).
    - Première version publique au début de l'année 1995.
    - "Personal Sommaire Page Tools" (ou "Personal Home Page"). Quelques utilitaires alors 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 : 7 millions de sites et +++ aujourd'hui

Introduction

- PHP utilise le moteur d'analyse 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 ").

Introduction

- Principaux sites d'informations sur php

http://www.php.net

http://www.zend.com

PHP

Intégration dans le document

Intégration dans le document

- Avant propos
  - 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 la plus utilisée.

- Un même document peut mèler du code XHTML et PHP, mais pour que le PHP soit exécuté, il doit porter l’extension voulue (cf ci-dessus)

Intégration dans le document

- Autrement dit :
  - 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 ("<p>Hello World</p>");
?>
```

Intégration dans le document

- **Le client ne reçoit que le résultat du script, (non le code qui a produit ce résultat)** Si l'on affiche le code source reçu dans un navigateur, pour l 'exemple précédent :

```
...
<body>
<p>Hello World</p>
</body>
...
```

Intégration dans le document

- **Séparateur d ’instruction**

;

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

```
<?php
/* Mes commentaires tiennent sur
plusieurs lignes */
// Un commentaire sur une ligne
?>
```

PHP

Types de données

Types de données

- **Types scalaires :**
  - booléen
  - entier
  - nombre à virgule flottante (float, double)
  - chaîne de caractères
- **Types composés :**
  - tableau
  - objet
- **Types spéciaux :**
  - ressource
  - null

Types de données

- **Booléens**
  - Valeurs true ou false

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

Types de données

- **Entiers**

en base :

  - décimale (base 10)
  - hexadécimale (base 16) - préfixés avec 0x
  - octale (base 8) - préfixés avec un 0

- Les entiers négatifs sont déclarés préfixés avec le signe moins (-)

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

Types de données

- **Nombres à virgule flottante (float, double)**

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

Types de données

- **Les chaînes de caractères**
  - 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';
?>
```

Types de données

- **Les chaînes de caractères**
  - 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)";
?>
```

Types de données

- **Les chaînes de caractères**
  - 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);
?>
```

Types de données

- **Les chaînes de caractères**
  - syntaxe Here Doc ("documentation ici")

```
<?php
// 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;
// $v sera remplacé par Toto
?>
```

Types de données

- **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, ...)

Types de données

- **Les tableaux**
  - 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.

Types de données

- **Les tableaux - exemple**

```
<?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
?>
```

Types de données

- **Les tableaux - exemples**

```
<?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...)
?>
```

Types de données

- **Les tableaux**
  - 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.

Types de données

- **Les tableaux**
  - **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
```

Types de données

- **Les tableaux multidimensionnels**

```
<?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"));
?>
```

Types de données

- **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();
?>
```

Types de données

- **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

Types de données

- **NULL**
  - NULL = **absence de valeur**
  - Constante, insensible à la casse.

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

  - Remarque : la fonction unset permet de détruire une variable

PHP

Variables

Variables

- **Un signe dollar "$" suivi du nom de la variable.**
  - Attention : le nom est sensible à la casse
  - 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** (voir diapositive suivante)

Variables

- PHP permet d'assigner les valeurs aux variables **par référence (assignation 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).
- **Avec un & (ET commercial) au début de la variable**

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

Variables

- **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)

Variables

- **Portée des variables**

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

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

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

$v n'a pas été assignée préalablement dans la fonction.

Variables

- **Portée des variables**

  != 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 $v1 et $v2 sont déclarées globales dans la fonction somme() : toutes les références à ces variables concerneront les variables globales.

Variables

- **Variables "static"**
  - portée locale uniquement
  - **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.

Variables

- **Variables "static"**
  - 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

Variables

- **Variables "static"**

  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 />";
}
?>
```

Variables

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

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

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

Variables

- 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>
```

Variables

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

ou

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

Variables

- Variables externes à PHP & Formulaires HTML / XHTML (GET et POST) & URL
  - Pour accéder à la valeur de « nom » en PHP
  - Avant PHP 4.1.0 (pour mémoire)
    - $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)

Variables

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

Variables

- **Variables prédéfinies**
  - Dépendent
    - du serveur
    - de la version du serveur
    - de la configuration du serveur
    - d'autres facteurs
  - Pour obtenir une liste des variables prédéfinies dans le contexte : **fonction phpinfo()**.
  - Voir la documentation PHP.

PHP

Constantes

Constantes

- **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**
    - **on ne préfixe pas le nom de la constante avec $.**
    - Par **convention**, les noms de constantes sont en **majuscules.**
    - Les constantes ont une **portée globale**.
  - Syntaxe

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

PHP

Opérateurs

Opérateurs

- Les opérateurs arithmétiques

\+ Addition d'opérandes

\- 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

Opérateurs

- Les opérateurs d'assignement

= Assigne la valeur de l'opérande droit à l'opérande 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

Opérateurs

- Les opérateurs sur les bits (opérateurs bitwise)

& 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).

Opérateurs

- Les opérateurs de comparaison

== 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

Opérateurs

- L'opérateur de contrôle d'erreur

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

- L'opérateur d'exécution

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

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

Opérateurs

- Les opérateurs logiques

  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

Opérateurs

- Les opérateurs de chaînes

. Concaténation de chaînes de caractères

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

Opérateurs

- Opérateurs divers

$ Référence une variable

& Référence l'adresse d'une variable

-> Référence une méthode ou propriété de classe

=> Assigne un index à un élément de tableau

Opérateurs

- Priorité des principaux opérateurs

Du + élevé

! ~ ++ -- $

*** / %**

+ -

< <= > >=

=== == !=

&

^

|

&&

||

= += -= *= /= .= %= &= |= ^=

AND

XOR

OR

Au moins élevé

PHP

Structures de contrôle

Structures de contrôle

- **if**

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

- **if ... else**

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

- **if ... elseif**
  - Combinaison de if et de else.

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

Structures de contrôle

- **switch**

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

Structures de contrôle

- **while**

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

- **do ... while**

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

- **for**

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

Structures de contrôle

- **foreach**

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

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

Structures de contrôle

- **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;
  }
}
```

Structures de contrôle

- **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;
     }
   }
}
```

Structures de contrôle

- **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;
}
```

Structures de contrôle

- **continue**
  - 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;
}
```

Structures de contrôle

- **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.
- **include()**
  - idem
  - mais include peut être placé dans une condition, elle ne sera exécutée que si la condition est remplie

Structures de contrôle

- **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.
- **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.

PHP

Fonctions

Fonctions

- 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;
}
?>
```

Fonctions

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

Fonctions

- **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 ;
?>
```

Fonctions

- **Passage d'arguments par référence**
  - **Exemple avec passage par référence**

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

Fonctions

- **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.

Fonctions

- **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)

Fonctions

- **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');
?>
```

PHP

Classes & objets

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

Classes & objets avec PHP 4

- Avec PHP 4
  - les attributs et méthodes sont toujours publics.

  - **Déclaration d'une classe**

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

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

Classes & objets avec PHP 4

- 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
```

Classes & objets avec PHP 4

- **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;
}
?>
```

Classes & objets avec PHP 4

- 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");
   }
}
?>
```

Classes & objets avec PHP 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

Classes & objets avec PHP 4

- **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

Classes & objets avec PHP 4

- **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() ;
```

Classes & objets avec PHP 4

- **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.

Classes & objets avec PHP 5

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

- **Constructeurs**
  - 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()**

Classes & objets avec PHP 5

- **Destructeurs**
  - 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";
   }
}
?>
```

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

Classes & objets avec PHP 5

- **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 quel 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.

Classes & objets avec PHP 5

- **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.

Classes & objets avec PHP 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 ->

Classes & objets avec PHP 5

- **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 des propriétés statiques à travers l'objet en utilisant l'opérateur ->

Classes & objets avec PHP 5

- **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...;
     }
}?>
```

Classes & objets avec PHP 5

- **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.

Classes & objets avec PHP 5

- **Interfaces**
  - **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...;
   }
}?>
```

Classes & objets avec PHP 5

- **"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...;
   }
}
?>
```