Introduction à PHP

Ryan Cassel
cassel@limsi.fr

Université Paris XI

Sommaire

- Présentation du PHP
- La syntaxe de base
- Les types
- Les variables
- Les constantes
- Les expressions
- Les opérateurs
- Les structures de contrôle
- Les fonctions
- Exemples et exercices

Présentation

- PHP est un langage de scripts généraliste et Open Source, spécialement conçu pour le développement d'applications web.
- Langage proche de la syntaxe du langage c
- Exécuté côté serveur
- Fichier texte (extension .php)
- Utilisé pour fournir un comportement dynamique (côté serveur) de page web
- Le code source PHP n'est pas accessible par un client web.

Que peut faire PHP ?

- Langage de script côté serveur
- Langage de programmation en ligne de commande
- Ecrire des applications clientes graphiques (PHP-GTK : gtk.php.net)

Présentation

- Conçu pour faciliter la création de pages web dynamiques
- Le code PHP est inséré dans le code HTML au moyen d'une pseudo-balise :

```
<?      //code PHP      ?>
<?PHP   //code PHP      ?>
<?php   //code PHP      ?>
```

- D'autres alternatives :
  - ASP, JSP, CGI, Perl, …

Exemple

```
<html>

<head>
    <title> Exemple de script PHP </title>
</head>

<body>

    <h1>
                < ?php echo "Bonjour, je suis un script PHP"; ?>
    </h1>

</body>

</html>
```

Présentation

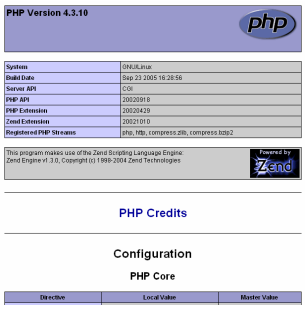
- vous écrivez une page HTML avec du code inclus à l'intérieur afin de réaliser une action précise (dans ce cas là, afficher du texte).
- le client ne reçoit que le résultat du script, sans aucun moyen d'avoir accès au code qui a produit ce résultat.
- Vous pouvez configurer votre serveur web afin qu'il analyse tous vos fichiers HTML comme des fichiers PHP. Ainsi, il n'y a aucun moyen de distinguer les pages qui sont produites dynamiquement des pages statiques.

Configuration

- Récupération des informations du système depuis PHP

```
<?php phpinfo(); ?>
```


phpinfo();



La syntaxe de base

- Passer du HTML au PHP
  - Lorsque PHP traite un fichier, il cherche les balises d'ouvertures et de fermetures, qui délimitent le code qu'il doit interpréter.

```
<p>Ceci sera ignoré.</p>

<?php echo 'Alors que ceci sera analysé par PHP.'; ?>

<p>Ceci sera également ignoré.</p>
```


La syntaxe de base

```
<?php
    if ($expression)
    {
?>
        <strong>Ceci est vrai.</strong>
<?php
    }
    else
    {
?>
        <strong>Ceci est faux.</strong>
<?php
    }
?>
```


La syntaxe de base

1.
```
<?php
        echo 'Si vous voulez réaliser des documents XHTML ou XML, faîtes
        comme ceci';
    ?>
```
2.
```
<script language="php">
        echo 'quelques éditeurs (comme FrontPage)
                n\'aiment pas ce genre d\'instructions';
    </script>
```
3.
```
<?
        echo 'ceci est le plus simple, une instruction SGML';
    ?>
    <?= expression ?> Ceci est la version courte pour "<? echo expression ?>"
```
4.
```
<%
        echo 'Vous pouvez optionnellement utiliser les balises ASP-style';
    %>
```

La syntaxe de base

- Le premier exemple est le plus communément utilisé et le plus recommandé.
- Pour réaliser du code portable, redistribuable, n'utilisez jamais les balises courtes.

Séparation des instructions

- Comme en C, PHP requiert que les instructions soient terminées par un point-virgule
- La balise fermante d'un bloc de code PHP implique automatiquement un point-virgule.
  Cependant nous vous recommandons de systématiquement terminer vos intructions par un point-virgule

```
<?php
   echo 'Ceci est un test';
?>

<?php echo 'Ceci est un test' ?>
```


Commentaires

- PHP supporte les commentaires 'C', 'C++' et Unix shell-style

```
<?php
echo 'Ceci est un test'; // Ceci est un commentaire sur une ligne, style c++

/* Ceci est un commentaire sur
        plusieurs lignes */

echo 'Ceci est un autre test';

echo 'Et un test final'; # Ceci est un commentaire style shell
?>
```


Commentaires

- Attention au commentaires de type C
- Il ne faut pas imbriquer les commentaires

```
<?php
    /*
      echo 'Ceci est un test'; /* Ce commentaire posera un problème */
    */
?>
```


Les types

- PHP supporte quatre types scalaires :
  - booléen
  - entier
  - nombre à virgule flottante
  - chaîne de caractères
- PHP supporte deux types composés :
  - tableau
  - objet
- PHP supporte deux types spéciaux

Les types

- Booléens
  - C'est le type le plus simple.
  - Un booléen exprime une valeur de vérité.
  - Il peut prendre comme valeur
    - soit **TRUE**
    - soit **FALSE**

Les types

- Booléens
  - Vous pouvez utiliser les mots-clés 'TRUE' et 'FALSE' pour spécifier une valeur de type booléen.
  - Ces mots-clés sont insensibles à la casse.

Les types

- Conversion en booléen
  - Pour explicitement convertir une valeur en booléen, utilisez les opérateurs de transtypage *(bool)* ou *(boolean)*.
  - Lors des conversions de valeurs de type bool, les valeurs suivantes sont considérées comme fausses (FALSE) :
    - Le booléen FALSE lui-même
    - L'entier 0 (zéro)
    - La chaîne de caractères 0.0 (zéro)
    - La chaîne de caractères vide et la chaîne de caractères "0"
    - Le tableau vide (aucun élément)
    - L'objet vide (aucun élément) (PHP 4 uniquement)
    - La type spéciale NULL
  - Toutes les autres valeurs sont considérées comme vraies (TRUE).

-1 est considéré comme vrai, tout comme les nombres autres que zéro (aussi bien positifs que négatifs) !

Les types

- Entiers
  - Un entier est un nombre de l'ensemble des entiers naturels Z : *Z = {...., -2, -1, 0, 1, 2, ...}*.
  - Les entiers peuvent être spécifiés en base décimale (dite aussi base 10), en hexadécimale (base 16) ou octale (base 8).
  - Les entiers peuvent être optionnellement précédés par le signe plus ou moins (- ou +).

```
<?php
    $a = 1234;      // nombre entier en base 10
    $a = -123;      // nombre entier négatif
    $a = 0123;      // nombre entier en base 8, octale (équivalent à 83 en base 10)
    $a = 0x1A;      // nombre entier en base 16, hexadécimale
                    // (équivalent à 26 en base 10)
?>
```


Les types

- Conversions en entier
  - Pour explicitement convertir une valeur en entier, utilisez les opérateurs de transtypage *(int)* ou *(integer)*.
  - Vous pouvez également convertir une valeur en entier avec la fonction **intval()**.
  - Lors de conversion entre un nombre décimal et un entier, le nombre sera arrondi à la valeur inférieure s'il est positif, et supérieure s'il est négatif (conversion dite 'vers zéro').

Ne transformez jamais une fraction inconnue en entier, car cela peut conduire à des résultats irrationnels.

```
<?php
    echo (int) ( (0.1+0.7) * 10 );
    // affiche 7!
?>
```


Les types

- Les nombres décimaux
  - Les nombres décimaux (connus aussi sous le vocable de "*double*", "*float*" ou "*nombre réels*") peuvent être spécifiés en utilisant la syntaxe suivante :

```
<?php
    $a = 1.234;
    $b = 1.2e3;
    $c = 7E-10;
?>
```


Les types

- Les chaines de caractères
  - Les chaînes de caractères sont des séquences de caractères.
  - En PHP, un caractère est un octet et il y en 256 de possibles.
  - Une chaîne peut être spécifiée de trois manières différentes :
    - guillemets simples
    - guillemets doubles
    - syntaxe HereDoc (pour plus de détails voir la doc PHP)

Note : La taille n'est pas un problème majeur pour une chaîne. Elle peut devenir très grande sans problème. Il n'y a pas à s'en faire pour cela.

Les types

- Les chaines de caractères
  - Guillemets simples
    - Le moyen le plus simple de spécifier une chaîne de caractères est d'utiliser les guillemets simples : '.
    - Pour spécifier un guillemet simple littéral, vous devez l'échapper avec un anti-slash (\).
    - Contrairement aux autres syntaxes, les variables présentes dans la chaîne ne seront PAS remplacées par leurs valeurs.

Les types

- Guillemets doubles
  - Si la chaîne est entourée de guillemets doubles ("), PHP va comprendre certaines séquences de caractères :

\n	Nouvelle ligne
\r	Retour à la ligne
\t	Tabulation horizontale
\\	Anti-slash
\$	Caractère $
\"	Guillemets doubles

Les types

- Les tableaux
  - Un tableau PHP est en fait une association ordonnée (littéralement, une *map*).
  - Une association est un type qui fait correspondre des valeurs à des *clés*

Les types

- Les tableaux

```php
<?php

  $arr = array("age" => "vieux", 12 => true);

  echo $arr["age"]; // vieux
  echo $arr[12];    // 1

?>
```


Les types

- Les tableaux
  - Tableaux associatifs

```php
<?php

  $arr = array("untableau" => array(6 => 5, 13 => 9, "a" => 42));

  echo $arr["untableau"][6];    // 5
  echo $arr["untableau"][13];   // 9
  echo $arr["untableau"]["a"];  // 42

?>
```


Les types

- Les tableaux
  - Indexation automatique

```php
<?php

  // Ce tableau est identique à
  array(5 => 43, 32, 56, "b" => 12);

  // Celui ci
  array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

Les types

- Les tableaux
  - La syntaxe à crochets

```
<?php
   $arr = array(5 => 1, 12 => 2);

   $arr[] = 56;          // Ceci revient à $arr[13] = 56;
                         // à ce moment du script

   $arr["x"] = 42;       // Ceci ajoute un nouvel élément
                         // avec l'index "x"

   unset($arr[5]);       // Ceci efface un élément du tableau

   unset($arr);          // Ceci efface tout le tableau
?>
```


Les types

- La valeur **NULL**
  - La valeur spéciale **NULL** représente l'absence de valeur. Une variable avec la valeur **NULL** n'a pas de valeur.
  - Une variable est considérée comme **NULL** si :
    - on lui a affecté la constante **NULL**.
    - elle n'a été définie à aucune valeur.
    - elle a été détruite par la fonction **unset()**.

Les types

- 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.
  - Conversion automatique

Les variables

- En PHP, les variables sont représentées par un signe dollar "$" suivi du nom de la variable.
- Le nom est sensible à la casse. *$x != $X*

Les variables

- Assignation
  - Simple :

```
$a = 10;
$b = $a;
```

  - Par référence

```
$a = 10;
$b = &$a;
```

  - Variables anonymes

```
$a = 10
$b = &(18*$a);
```


Les variables

- Les variables prédéfinies
  - PHP fourni un grand nombre de variables pré-définies. Exemple :
    - $GLOBALS
      - Contient une référence sur chaque variable qui est en fait disponible dans l'environnement d'exécution global. Les clés de ce tableau sont les noms des variables globales.
      - Voir phpinfo() pour la liste des variables globales.

Les variables

- Portée des variables
  - La portée d'une variable dépend du contexte dans lequel la variable est définie.
  - Lorsque vous définissez une fonction, la portée d'une variable définie dans cette fonction est locale à la fonction.

Les variables

Que fait ce script ?

Fichier toto.php

```
<?php
   $a = 1;
   include 'b.inc';
?>
```

Fichier b.inc

```
<?php
    echo $a;
?>
```


Les variables

```
<?php
   $a = 1;
   include 'b.inc';
?>
```

Ici, la variable *$a* sera accessible dans le script inclus b.inc.

Les variables

Que fait ce script ?

```
<?php
   $a = 1; /* portée globale */

   function test()
   {
      echo $a; /* portée locale */
   }

   test();
?>
```


Les variables

```
<?php
   $a = 1; /* portée globale */

   function test()
   {
      echo $a; /* portée locale */
   }

   test();
?>
```

Le script n'affiche rien car l'instruction echo() utilise la varialbe locale $a et celle-ci n'a pas été initialisée préalablement dans la fonction.

Les variables

- Le mot clé global

```
<?php
   $a = 1;
   $b = 2;

   function somme()
   {
      global $a, $b;
      $b = $a + $b;
   }

   somme();
   echo $b;
?>
```

Les variables

- Script équivalent avec le tableau associatif $GLOBALS

```
<?php
   $a = 1;
   $b = 2;

   function somme()
   {
      $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
   }

   somme();
   echo $b;
?>
```


Les variables

- Les variables static
  - Une variable statique a une portée locale uniquement, mais elle ne perd pas sa valeur lorsque le script appelle la fonction.

Les variables

- Les variables static

```
<?php
   function Compteur()
   {
      $a = 0;
      echo $a;
      $a++;
   }
?>
```

Cette fonction est un peu inutile car à chaque fois qu'elle est appelée, elle initialise *$a* à 0 et affiche "0".

Les variables

- Les variables static

```
<?php
   function Compteur()
   {
      static $a = 0;
      echo $a;
      $a++;
   }
?>
```

Maintenant, à chaque fois que la fonction Compteur() est appelée, elle affichera une valeur de *$a* incrémentée de 1.

Les variables

- Déclaration des variables static

```
<?php
   function foo()
   {
      static $int = 0;          // correct
      static $int = 1+2;        // faux  (car c'est une expression)
      static $int = sqrt(121);  // faux  (car c'est aussi une
   expression)

      $int++;
      echo $int;
   }
?>
```


Les variables

- Les variables dynamiques
  - Il est pratique d'avoir parfois des noms de variables qui sont variables. C'est-à-dire un nom de variable qui est affecté et utilisé dynamiquement.
  - Une variable dynamique prend la valeur d'une variable et l'utilise comme nom d'une autre variable.

Les variables

- Les variables dynamiques

```
<?php

   $a  = 'bonjour ';
   $$a = 'le monde';

?>
```


Les variables

- Les variables dynamiques

```
<?php
    echo "$a ${$a}";
?>
```

Est équivalent à :

```
<?php
    echo "$a $bonjour";
?>
```

Affichera :

bonjour le monde

Les constantes

- Une constante est un identifiant (un nom) qui représente une valeur simple.
- Comme son nom le suggère, cette valeur ne peut jamais être modifiée durant l'exécution du script (les constantes magiques __FILE__ et __LINE__ sont les seules exceptions).
- Le nom d'une constante est sensible à la casse, par défaut.
- Par convention, les constantes sont toujours en majuscules.

Les constantes

- Syntaxe
  - Vous pouvez définir une constante en utilisant la fonction **define()**.
  - Une fois qu'une constante est définie, elle ne peut jamais être modifiée, ou détruite.

Les constantes

- Seuls les types de données scalaires peuvent être placés dans une constante.

```
<?php
    // Noms valides
    define("FOO",    "something");
    define("FOO2",   "something else");
    define("FOO_BAR", "something more")

    // Noms invalides
    define("2FOO",   "something");

    // Ce nom est valide, mais évitez-le :
    // PHP peut un jour fournir une constante magique nommée
    // ainsi, ce qui va corrompre vos scripts.
    define("__FOO__", "something");
?>
```


Les constantes

- Il y a des différences entre les constantes et les variables :
  - Les constantes ne commencent pas par le signe ($);
  - Les constantes sont définies et accessibles à tout endroit du code, globalement.
  - Les constantes ne peuvent pas être redéfinies ou indéfinies une fois qu'elles ont été définies.
  - Les constantes ne peuvent contenir que des scalaires.

Les constantes

- Lorsque vous utilisez une constante non définie, PHP suppose que vous utilisez le nom de la constante.
- Une note sera générée.
- Utilisez la fonction **defined()** pour savoir si une constante existe ou pas.

Les constantes

```
<?php
  define("CONSTANTE", "Bonjour le monde.");
  echo CONSTANTE; // affiche "Bonjour le monde."
  echo Constante; // affiche "Constante" et une note.
?>
```


Les constantes

- Les constantes magiques :
  - **__LINE__** La ligne courante dans le fichier.
  - **__FILE__** Le chemin complet et le nom du fichier courant. Si utilisé dans un include, le nom du fichier inclus est retourné. Depuis PHP 4.0.2, **__FILE__** contient toujours le chemin absolu alors que les anciennes versions contenaient le chemin relatif, dans certaines circonstances.
  - **__FUNCTION__** Le nom de la fonction. (Ajouté en PHP 4.3.0) Depuis PHP 5, cette constante retourne le nom de la fonction comme il a été déclaré (sensible à la casse). En PHP 4, cette valeur est toujours en minuscule.

Les expressions

- La manière la plus simple de définir une expression est : "tout ce qui a une valeur".
- Les formes les plus simples d'expressions sont les constantes et les variables.

Les opérateurs

- Il y a trois types d'opérateurs :
  - Les opérateurs unaires, qui opèrent sur une seule valeur, par exemple ! (l'opérateur de négation) ou ++ (l'opérateur d'incrémentation).
  - Les opérateurs binaires ; ce groupe contient la plus part des opérateurs supportés par PHP.
  - Les opérateurs de terminaison : ?:. Ils doivent être utilisés pour choisir entre deux expressions dépendants d'une troisième.

Les opérateurs

- Les opérateurs arithmétiques
  - \+ addition
  - \- soustraction
  - * multiplication
  - / division
  - % modulo

Les opérateurs

- Les opérateurs d'assignation

```
<?php
    $a = ($b = 4) + 5;
    // $a est maintenant égal à 9, et $b vaut 4.
?>

<?php
    $a = 3;
    $a += 5; // affecte la valeur 8 à la variable $a correspond à l'instruction '$a = $a + 5';
    $b = "Bonjour ";
    $b .= " tout le monde!";  // affecte la valeur "Bonjour tout le monde!" à
                              //  la variable $b
                              //  identique à $b = $b." tout le monde!";

?>
```


Les opérateurs

- Les opérateurs de comparaison

$a == $b	Egal	**TRUE** si *$a* est égal à *$b*.
$a === $b	Identique	**TRUE** si *$a* est égal à *$b* et qu'ils sont de même type (introduit en PHP 4).
$a != $b	Différent	**TRUE** si *$a* est différent de *$b*.
$a <> $b	Différent	**TRUE** si *$a* est différent de *$b*.
$a !== $b	Différent	**TRUE** si *$a* est différent de *$b* ou bien qu'ils ne sont pas du même type. (introduit en PHP 4)
$a < $b	Plus petit que	**TRUE** si *$a* est strictement plus petit que *$b*.
$a > $b	Plus grand	**TRUE** si *$a* est strictement plus grand que *$b*.
$a <= $b	Inférieur ou égal	**TRUE** si *$a* est plus petit ou égal à *$b*.
$a >= $b	Supérieur ou égal	**TRUE** si *$a* est plus grand ou égal à *$b*.

Les opérateurs

- Les opérateurs de contrôle d'erreur
  - PHP supporte un opérateur de contrôle d'erreur : c'est @.
  - Lorsque cet opérateur est ajouté en préfixe d'une expression PHP, les messages d'erreur qui peuvent être générés par cette expression seront ignorés.

Les opérateurs

- Les opérateurs de contrôle d'erreur
  - Si l'option *track_errors* est activée, les messages d'erreurs générés par une expression seront sauvés dans la variable globale *$php_errormsg*.
  - Cette variable sera écrasée à chaque erreur.

Les opérateurs

- Les opérateurs de contrôle d'erreur
  - L'opérateur "@" va aussi désactiver les rapports d'erreurs critiques, qui stoppent l'exécution du script.
  - Si vous utilisez "@" pour supprimer les erreurs de certaines fonctions, et que cette fonction n'existe pas, ou qu'elle a été mal orthographiée, vous n'aurez aucune indication.

Les opérateurs

- Opérateurs d'incrémentation et décrémentation
  - PHP supporte les opérateurs de pre- et post-incrémentation et décrémentation, comme en langage C.

Les opérateurs

- Opérateurs d'incrémentation et décrémentation

++$a	Pre-incrémente	Incrémente *$a* de 1, puis retourne *$a*.
$a++	Post-incrémente	Retourne *$a*, puis l'incrémente de 1.
--$a	Pré-décrémente	Décrémente *$a* de 1, puis retourne *$a*.
$a--	Post-décrémente	Retourne *$a*, puis décrémente *$a* de 1.

Les opérateurs

- Opérateurs logiques

$a and $b	ET (*And*)	Vrai si *$a* ET *$b* sont vrais
$a or $b	OU (*Or*)	Vrai si *$a* OU *$b* est vrai
$a xor $b	XOR (*Xor*)	Vrai si *$a* OU *$b* est vrai, mais pas les deux en même temps.
! $a	NON (*Not*)	Vrai si *$a* est faux.
$a && $b	ET (*And*)	Vrai si *$a* ET *$b* sont vrais.
$a || $b	OU (*Or*)	Vrai si *$a* OU *$b* est vrai.

Les opérateurs

- Opérateurs de chaînes
  - Il y a deux opérateurs de chaînes de caractères.
    - Le premier est l'opérateur de concaténation ('.'), qui retourne la concaténation de ses deux arguments.
    - Le second est l'opérateur d'assignation concaténant (.=).

Les opérateurs

- Opérateurs de chaînes

```php
<?php
   $a = "Bonjour ";
   $b = $a . "le monde!"; // $b contient "Bonjour le monde!"

   $a = "Bonjour ";
   $a .= "Monde!"; // $a contient "Bonjour Monde!"
?>
```


Les opérateurs

- Opérateurs d'exécution
  - PHP supporte un opérateur d'exécution : guillemets obliques ("``"). Notez bien qu'il ne s'agit pas de guillemets simples.
  - PHP essaiera d'exécuter le contenu de ces guillemets obliques comme une commande shell.
  - Le résultat sera retourné (i.e. : il ne sera pas simplement envoyé à la sortie standard, il peut être assigné à une variable).
  - Utilisez les guillemets obliques revient à utiliser la fonction **shell_exec()**.

Les opérateurs

- Opérateurs d'exécution

```php
<?php
    $output = `ls -al`;
    echo "<pre>$output</pre>";
?>
```

Les structures de contrôle

- Tous les scripts PHP sont une suite d'instructions.
- Une instruction peut être une assignation, un appel de fonction, une instruction conditionnelle ou bien une instruction qui ne fait rien (une instruction vide).
- Une instruction se termine habituellement par un point virgule (";").
- Plusieurs instructions peuvent être regroupées en bloc, délimité par des accolades ("{}").
- Un bloc est considéré comme une instruction.

Les structures de contrôle

- **if else**
  - Les fonctionnalités de l'instruction *if* et *else* sont les mêmes en PHP qu'en C

```
<?php
   if (expression)
   {
     instructions;
   }
   else
   {
     instructions;
   }
?>
```


Les structures de contrôle

- elseif
  - Revient à faire : else if

Les structures de contrôle

- while
  - Cette boucle se comporte de la même manière qu'en C

```
<?php
   while (expression)
   {
         instructions;
   }
?>
```

  - PHP exécute l'instruction tant que l'expression de la boucle *while* est évaluée comme **TRUE**.

Les structures de contrôle

- do-while
  - Semblable au while.

```
<?php
   $i = 0;
   do {
     echo $i;
   } while ($i > 0);
?>
```


Les structures de contrôle

- for
  - Comme les boucles *for* du langage C.

```
<?php
   for (expr1; expr2; expr3)
   {
         instructions;
   }
?>
```

- La première expression (*expr1*) est évaluée (exécutée), quoi qu'il arrive au début de la boucle.
- Au début de chaque itération, l'expression *expr2* est évaluée. Si l'évaluation vaut **TRUE**, la boucle continue et l'instruction est exécutée. Si l'évaluation vaut **FALSE**, l'exécution de la boucle s'arrête.
- A la fin de chaque itération, l'expression *expr3* est évaluée (exécutée).

Les structures de contrôle

- foreach
  - PHP 4 introduit une commande *foreach*, comme en Perl ou d'autres langages.
  - C'est un moyen simple de passer en revue un tableau.

```
<?php
   $arr = array(1, 2, 3, 4);
   foreach ($arr as &$value) {
     $value = $value * 2;
   }
   // $arr vaut maintenant array(2, 4, 6, 8)
?>
```


Les structures de contrôle

- break
  - L'instruction *break* permet de sortir d'une structure *for*, *foreach*, *while*, *do-while* ou *switch*.

Les structures de contrôle

- switch
  - L'instruction *switch* équivaut à une série d'instructions *if*.
  - En de nombreuses occasions, vous aurez besoin de comparer la même variable (ou expression) avec un grand nombre de valeurs différentes, et d'exécuter différentes parties de code suivant la valeur à laquelle elle est égale.

Les structures de contrôle

- switch

```
<?php
   switch ($i) {
   case "tarte":
     echo "i est une tarte";
     break;
   case "barre":
     echo "i est une barre";
     break;
   case "gateau":
     echo "i est un gateau";
     break;
   }
?>
```


Les structures de contrôle

- return()
  - Si appelée depuis une fonction, la commande return() termine immédiatement la fonction, et retourne l'argument qui lui est passé.
  - Si appelée depuis l'environnement global, l'exécution du script est interrompue.
  - Si le script courant était include() ou require(), alors le contrôle est rendu au script appelant, et la valeur retournée sera utilisée comme résultat de la fonction include().
  - Si return() est appelée depuis le script principal, alors l'exécution du script s'arrête.

Les structures de contrôle

- require()
  - require() inclut et exécute un fichier PHP.
  - La commande require() se remplace elle-même par le contenu du fichier spécifié, comme la fonction include().
  - require() et include() sont identiques, sauf dans leur façon de gérer les erreurs.
    - include() produite une Alerte (*warning*)
    - require() génère une erreur fatale.

Les structures de contrôle

- include()
  - La fonction include() inclut et exécute le fichier spécifié en argument.

Les fonctions

- Une fonction peut être définie en utilisant la syntaxe suivante :

```
<?php
    function foo($arg_1, $arg_2, /* ..., */ $arg_n)
    {
        echo "Exemple de fonction.\n";
        return $retval;
    }
?>
```

- Elle sera appelée en utilisant la syntaxe suivante:

```
<?php
    foo($arg_1, $arg_2, /* ..., */ $arg_n);
?>
```


Les fonctions

- **Fonctions internes**
  - PHP dispose de nombreuses fonctions et structures standards.
    - phpinfo()
    - …

Les fonctions

- Les fonctions sur les fichiers
  - La plupart des fonctions du langage C sont disponibles :
    - fopen()
    - fclose()
    - fgetc()
    - fgets()
    - …
  - Des fonctions spécifiques au PHP facilite l'accès aux fichiers :
    - readfile()
    - opendir()
    - readdir()
    - …

Exercice1

- Créez une fonction qui affiche une pyramide de n lignes

```
.....o.....   1
....ooo....   2
...ooooo...   3
..ooooooo..   4
.ooooooooo.   5
ooooooooooo   6
```


Correction

```
<?PHP
    function pyramid($n)
    {
        $nb_espace = $n;
        $nb_point = 1;

        while($nb_espace>=0)
        {
            for($i = 0; $i < $nb_espace; $i++)
                echo ".";
            for($i = 0; $i < $nb_point; $i++)
                echo "o";
            for($i = 0; $i < $nb_espace; $i++)
                echo ".";

            echo "<br />";

            $nb_espace--;
            $nb_point++;
        }
    }
?>
```

Exercice 2

- Ecrire une fonction qui calcule la factorielle de *n*. *n!*
  - *1! = 1*
  - *n! = n x (n – 1)!*

Correction

- Fonction itérative :

```
<?PHP

function facto($n)
{
    $resultat = 1;
    while($n>1)
    {
        $resultat *= $n;
        $n--;
    }
    return($resultat);
}

?>
```


Correction

- Fonction récursives :

```
<?PHP

function facto($n)
{
    if($n == 1)
        return 1;
    else
        return(n*facto($n-1));
}

?>
```


Exercice 3

- Créer une fonction qui affiche la liste des fichiers du répertoire courant.

Correction

- Méthode intuitive

```
<?php

function AfficheFichiers($chemin)
{
   chdir(($chemin);
   echo `ls -l`; \\ ou shell_exec('ls –l');
}
?>
```


Correction

- Méthode simple

```
<?php

function AfficheFichiers($chemin)
{
   if ($handle = opendir($chemin))
   {
      while (false !== ($file = readdir($handle)))
         echo "$file\n";
   }
   closedir($handle);
}
?>
```

Exercice 4

- Créez une fonction qui remplit un tableau avec les noms de fichiers d'un dossier

Correction

```
<?php
function ListeFichier($chemin)
{
    $handle = opendir($chemin);

    while(false !== ($file = readdir($handle)) )
    {
        $files[] = $file;
    }

    closedir($handle);

    return($files);
}
?>
```


Correction

```
<?php
function ListeFichier($chemin)
{
  $files = scandir($chemin)
  return($files);
}
?>
```


Références sur le net

- www.php.net