le cnam

LES TECHNOLOGIES DU WEB

1ère Partie : **Introduction au Web**

- 1- **Introduction à l'Hypertexte**
- 2- **Présentation du protocole HTTP**
- 3- **Principes de bases des CGI**
- 4- **Présentation du WEB2 (AJAX)**

2ème Partie : **Présentation de HTML & XHTML**

3ème Partie : **Présentation de Javascript**

4ème Partie : Introduction à PHP

5ème Partie : **Introduction à XML & XSLT**

le cnam

INITIATION à PHP

NOTIONS DE BASE en programmation Web avec PHP

le cnam

Initiation à PHP - PLAN

- Introduction
- Variables et constantes
- Opérateurs
- Tableaux et tableaux associatifs
- Structures de contrôles
- Fonctions
- Le système de fichiers
- Programmation modulaire (OO)
- Accès aux bases de données
- Débuguer un script PHP
- Des conseils de programmation
- Des comparaisons entre PHP et ASP
- Interaction avec AJAX

le cnam

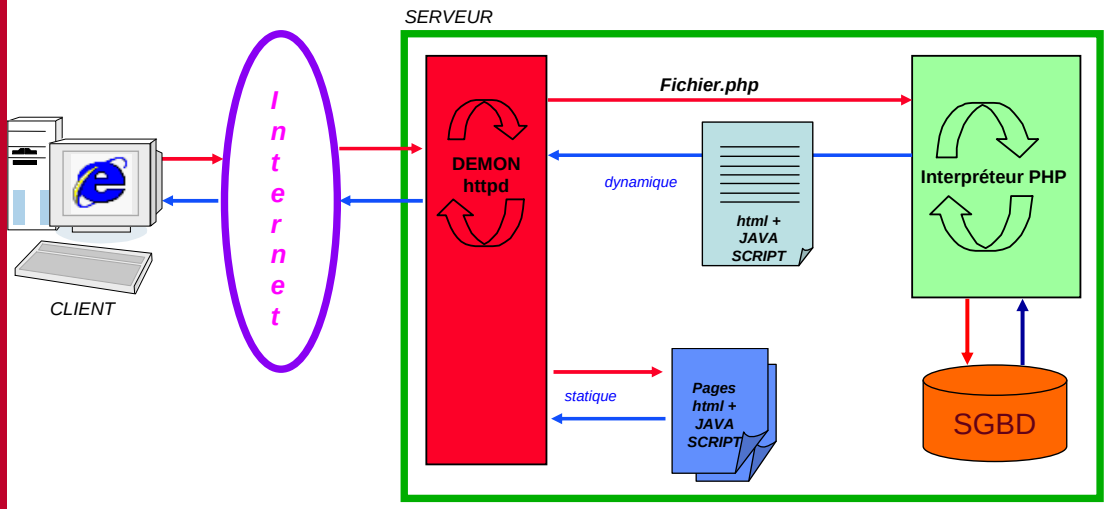
INTRODUCTION

PHP est un langage interprété orienté Web. Syntaxiquement, c'est un mélange de C et de Perl. Les scripts PHP sont lus et interprétés par le moteur PHP.

PHP comporte plus de 500 fonctions. Il est fournit avec des librairies offrant des fonctionnalités diverses :

- ✓ accès aux bases de données,
- ✓ fonctions d'images,
- ✓ sockets,
- ✓ protocoles Internet divers...

PRINCIPE de FONCTIONNEMENT



Lorsqu'une requête HTTP est soumise au serveur Web pour une page dont l'extension est «.php», comme pour un fichier HTML, le serveur commence par rechercher dans son arborescence le fichier d'extension «.php». Il va ensuite passer la main à un sous-processus (une dll bien particulière) qui va interpréter le script PHP et produire dynamiquement du code HTML. Ce code HTML est alors envoyé au travers du réseau au navigateur client. De plus, aucune ligne de code PHP n'apparaît côté client dans la mesure où tout le code a été interprété.

SYNOPSIS

Un script PHP peut comprendre a la fois du code PHP et du code HTML, non interprété. On doit donc encadrer les parties comportant le code PHP entre 2 balises **<?** et **?>**. Le reste de la page n'est pas interprété.

```
<html><head><title>
<? $titrepage = "Mon premier script PHP";
    echo $titrepage; ?>
</title></head><body>
<h1><? echo $titrepage ?></h1>
<?    echo " <b> Hello, World ! </b>"; ?>
</body></html>
```

Note: La balise **<?php** est équivalente a **<?**. On peut également utiliser les balises **<script language="php">** et **</script>** Enfin, pour les programmeurs ASP, sachez que les balises **<%** et **%>** sont également reconnues.

AFFICHER DU TEXTE (1)

Le séparateur d'instructions est le ;
Il est obligatoire, sauf si l'instruction est suivie de la balise ?>

La fonction echo affiche un (ou plus) argument. Si l'argument est une chaîne entre simple quotte ' il est affiché tel quel.

```
echo 'Hello, World';
```

Avec le quotte double " les variables contenues dans cette chaîne sont interprétées.

```
$nom= "Toto";
echo "Hello, $nom"; // Hello, Toto
echo 'Hello, $nom'; // Hello, $nom
```


AFFICHER DU TEXTE (2)

On peut également inclure le résultat d'une fonction directement dans un echo.

```
echo "Votre Nom en majuscule : ", strtoupper( "Toto" ), "\n";
// la fonction strtoupper mets tous les caractères de la chaîne en majuscule.
```

Pour afficher le caractère " , on l'insère à l'aide du caractère d'échappement \

```
echo " Escaping de caractères : \" \n";
```

On peut inclure des caractères spéciaux pour contrôler le flux affiché :

- \n saut de ligne
- \r fin de ligne
- \t tabulation

Pour terminer l'exécution du script, on utilise la fonction exit();

AFFICHER DU TEXTE (3)

Pour commenter le code, on utilise :

Commentaire sur une ligne: ***//*** **ou** ***#***
Commentaire sur plusieurs lignes: ***/* ... */***

Utilisation en mode ligne de commande :

On peut exécuter un script PHP en ligne de commande, ce qui permet des usages hors du simple cadre "Web".

L'option **-q** évite l'affichage de la première ligne

Content-type: text/html

C:\WEB\PHP\> php -q monscript.PHP

LES VARIABLES (1)

Visibilité et affectation

PHP n'est pas un langage fortement structuré, il ne contient donc pas de partie déclarative clairement définie. Pour définir une variable, il suffit de l'initialiser.

Les variables sont précédées du signe $, quelque soit leur type. Ainsi pour déclarer une variable var :

```
$var=1;
```

La variable **$var** est alors définie et vaut 1. Elle devient immédiatement accessible et ce jusqu'à la fin du script.

LES VARIABLES (2)

Type de variables

Les variables PHP sont a typage faible. C'est PHP qui décide de son type lors de l'affectation. Il existe six types de données :

- ✓ Entier *(int, integer)*
- ✓ Décimal *(real, float, double)*
- ✓ Chaîne de caractères *(string)*
- ✓ Tableau *(array)*
- ✓ Objet *(object)*
- ✓ Booléen *(boolean, uniquement PHP4)*

Il est parfois utile de forcer le type d'une variable. On utilise la fonction settype ou bien les opérateurs de casting (int), (string) settype renvoie vrai si la conversion a fonctionné, faux sinon.

```
$a= 3.1415;
$result= settype( $a, "integer" ); // => $a = 3  , $result = 1
```


LES VARIABLES (3)

Les opérateurs de conversion sont :

- ✓ (string) conversion en chaîne de caractères
- ✓ (int) conversion en entier, synonyme de (integer)
- ✓ (real) conversion en double, synonyme de (double) et (float)
- ✓ (array) conversion en tableau
- ✓ (object) conversion en objet
- ✓ (bool) conversion en booléen

```
$var= 1;          // $var est de type "integer" et vaut 1.
$chn=(string) $var ; // $var est de type "string" et vaut " 1 ".
```

On peut également utiliser strval, intval, doubleval qui renvoient la variable convertie en chaîne / entier / réel.

```
$strPI= "3.1415";
$intPI= intval( $strPI );
$PI= doubleval( $strPI );
echo " $strPI / $intPI / $PI";          // => 3.1415 / 3 / 3.1415
```

Remarque : Ces fonctions ne fonctionnent pas sur les tableaux.

LES VARIABLES (4)

Règles des conversions implicites :

- ✓ Si la chaîne de caractères contient un *point*, un *e* ou un *E* ainsi que des *caractères numériques*, elle est convertie en décimal,
- ✓ Si la chaîne de caractères ne contient que des *caractères numériques*, elle est convertie en entier,
- ✓ Si la chaîne de caractères est composée de *chiffres* et de *lettres*, elle est convertie en entier et vaut 0,
- ✓ Si la chaîne de caractères contient *plusieurs mots*, seul le premier est pris en compte et est converti selon les règles ci-dessus.

```
$var1 = 1;          // $var1 est de type "integer" et vaut 1.
$var2 = 12.0;       // $var2 est de type "double" et vaut 12.
$var3 = "PHP";      // $var3 est de type "string" et vaut "PHP".
$var4 =  false;     // $var4 est de type "boolean" et vaut false.
$var5 = "5a";       // $var5 est de type "string" et vaut "5a".
```


LES VARIABLES (5)

Références

PHP4 permet d'exploiter les références aux variables, à l'instar du langage C. Une référence à une variable est un accès à la zone mémoire qui contient la valeur de cette variable.

Cette référence est désignée par le caractère & placé devant le nom de la variable.

```
$a = 1 ;            // $a a pour valeur 1.
$b = &$a ;
// $b et $a pointent sur la même zone mémoire.
// Ce sont donc deux noms pour la même variable.
echo  " $a, $b " ;  // Affiche 1, 1
$a = 2 ;
echo  " $a, $b " ;  // Affiche 2, 2
```

LES VARIABLES (6)

Tests sur les variables

La fonction isset permet de tester si une variable est définie.

La fonction unset permet de supprimer la variable, et de désallouer la mémoire utilisée.

```
echo isset($a);  // => 0 (faux)
$a= " ";
unset($a);       // => 1 (vrai)
echo isset($a);  // => 0 (faux)
```

La fonction gettype permet de connaître le type de la variable. Elle renvoie une chaîne : "string" ou "integer" ou "double" ou "array" ou "object".

```
$a= 12;
echo gettype($a) ; // => "integer"
$a= $a / 10;
echo gettype($a) ; // => "double"
unset($a);
echo gettype($a) ; // => "string«
```

Remarque : Si la variable n'est pas définie, elle renvoie "string".

LES VARIABLES (7)

Tests sur les variables (suite et fin)

On peut également tester un type particulier a l'aide des fonctions is_array, is_string, is_int, is_float, is_object .

```
$a= 123;
echo is_int($a); // => (vrai)
echo is_double($a) // => (faux)
echo is_string($a) // => (faux)
$a += 0.5;
echo is_float($a) // => (vrai)
```

Remarque :

- *Les fonctions is_double et id_real sont équivalentes à is_float.*
- *Les fonctions is_long et is_integer sont équivalentes à is_int.*

LES CONSTANTES

PHP permet de définir des constantes a l'aide de la fonction **define**.

```
define("CONSTANTE", "rouge" );
```

Deux constantes sont prédéfinies par PHP :

- ✓ __FILE__ **contient le nom du fichier,**
- ✓ **et** __LINE__ **le numéro de la ligne courante.**

```
define( "NEXTPAGE", "script2.PHP" );
echo "Page courante : ", __FILE__ , "Page suivante : ",
   NEXTPAGE;
```

➔ **pas de $ pour des constantes.**

LES OPERATEURS (1)

PHP dispose des opérateurs classiques inspirés des langages C et Perl.

Comparaison

==	égalité
>	inférieur strict
<	supérieur strict
<=	inférieur ou égal
>=	supérieur ou égal
!=	négation

Logiques

Les opérateurs logiques sont utilisés dans les tests, par exemple dans un « if (condition) »

&&	et
||	ou
xor	ou exclusif
!	négation

Remarque : les opérateurs and, or , not sont également disponibles et font la même chose.

LES OPERATEURS (2)

Arithmétiques

+	addition
-	soustraction
/	division
*	multiplication
%	modulo
++	incrément
--	décrément

Remarque : l'opérateur / renvoie un entier si les 2 opérandes sont des entiers, sinon il renvoie un flottant.

Affectation

=	affectation
+=	addition puis affectation
-=	soustraction puis affectation
*=	multiplication puis affectation
/=	division puis affectation
%=	modulo puis affectation

```
$n = 0;
$n += 2;          // $n vaut 2
$n *= 6;          // $n vaut 12
$r= $n % 5;     // 12 modulo 5 => $r = 2
if( ++$n == 13 ) echo " pas de chance ";
                  // pré-incrément  le test renvoie vrai
```


LES OPERATEURS (3)

Binaires

&	ET	echo 3 & 6 ;	// 0011 AND 0110 => 2
|	OU	echo 3 | 6 ;	// 0011 OR 0110 => 7
^	XOR	echo 3 ^ 6 ;	// 0011 XOR 0110 => 5
~	NOT	echo ~3;	// NOT 3 => -4

Divers

- L'opérateur de concaténation est utilisable sur les chaînes scalaires.

```
$chaîne = "Votre nom est" ;
$nom = "Toto";
echo  $chaîne . " " . $nom;        // affiche "Votre nom est Toto"
```

- L'opérateur ? : ou *opérateur de test trinaire*. Sa syntaxe est [test logique] ? [expression si vrai] : [expression si faux]

```
$a= $b =1;
( $a == $b ) ? $c= 10 : $c = 20;   // effectue $c = 10;
```

- On peut également l'utiliser pour compacter les séquence de test / affectations

```
$réponse = ( $a == $b ) ? "a égal b" : "a différent de b" ;
echo $réponse; // affiche "a égal b" car le test ( $a == $b ) renvoie vrai
```

LES TABLEAUX (1)

Déclarations :

```
$fruits= array();
```

Affectations :

```
$fruits[0]= "pomme";
$fruits[1]= "banane";
$fruits[] .= "orange"; // équivaut a $fruits[2]= "orange"
$fruits= array( "pomme", "banane", "orange" );
```

Fonctions relatives :

sizeoff : Renvoie le nombre d'éléments d'un tableau. C'est un équivalent de count.

```
$nbelements= sizeof( $tableau );
```

is_array : renvoie true si la variable est de type tableau (ou tableau associatif), false sinon.

reset : la fonction reset($tableau) place le pointeur interne sur le premier élément du tableau, chaque variable tableau possède un pointeur interne repérant l'élément courant.

LES TABLEAUX (2)

Fonctions relatives (suite):

end : la fonction end($tableau) place le pointeur interne du tableau sur le dernier élément du tableau.

current : renvoie l'élément courant du tableau.

next : déplace le pointeur vers l'élément suivant, et renvoie cet élément. S'il n'existe pas, la fonction renvoie false.

prev : déplace le pointeur vers l'élément précédent, et renvoie cet élément. S'il n'existe pas, la fonction renvoie false.

each : la fonction $a=each($tablo) renvoie l'index et la valeur courante dans un tableau à 2 elements, $a[0] contient l'index, $a[1] la valeur.

list : la fonction list($scalar1, $scalar2, ...) construit un tableau temporaire à partir des variables scalaires passées en argument.

key : la fonction key($tablo) renvoie l'index de l'élément courant du tableau.

LES TABLEAUX (3)

Fonctions relatives (suite et fin):

sort, rsort, usort, uasort : sont différentes fonctions de tri de tableau.

sort trie par valeurs croissantes, *rsort* par valeurs décroissantes

```
$tableau_trie = sort( $tableau );
```

usort et *uasor*t permettent au programmeur d'implémenter lui-même la fonction de tri utilisée. PHP appelle successivement la fonction qui doit retourner -1 / 0 / 1 suivant que le premier élément est inférieur / égal / supérieur au second. Dans l'exemple ci-dessous, on implémente un tri qui ne tient pas compte des majuscules/ minuscules

```
function compare_maj( $elem1, $elem2 ) {
if( strtoupper( $elem1 ) == strtoupper( $elem2 ) ) return 0;
return ( strtoupper( $elem1 ) < strtoupper( $elem2 )) ? -1 : 1;
}
……..
$tableau_trie = usort( $tableau, "compare_maj" );
```


LES TABLEAUX ASSOCIATIFS (1)

Un tableau associatif est un tableau dont l'index est une chaîne de caractère au lieu d'un nombre. On parle aussi de "hash array" ou "hash". Il se déclare comme un tableau traditionnel, la distinction se fait lors de l'affectation.

Déclarations :

```
$calories= array();          // comme un tableau
```

Affectations :

Affectons un nombre de calories moyen aux fruits.

```
$calories["pommes"]= 300;
$calories["banane"]= 130;
$calories["litchie"]= 30;
```

LES TABLEAUX ASSOCIATIFS (2)

Fonctions relatives :

isset : pour tester l'existence d'un élément, on utilise la fonction *isset()* .

```
if( isset( $calories["pommes"] ) ) {
echo "une pomme contient ", $calories["pommes"] , " calories\n";
} else {
echo "pas de calories définies pour la pomme\n";
}
```

asort, arsort, ksort, aksort : Ces fonctions de tri conservent la relation entre l'index et la valeur, généralement le cas dans un tableau associatif.

- ✓ *asort* trie par valeurs croissantes,
- ✓ *arsort* par valeurs décroissantes,
- ✓ ksort trie par index (key) croissantes.

LES STRUCTURES DE CONTRÔLES (1)

Les tests IF

Syntaxes :

Test if " basique " :

```
if( [condition] ) {
   ...
}
```

Test if-else :

```
if( [condition]  ) {
   ...
} else {
   ...
}
```

Test if-elseif :

```
if( [condition]  ) {
   ...
} elseif( [condition] )
  {
   ...
  }
```

Dans le cas de plusieurs tests successif portant sur une Même variable, on utilisera plutôt le test switch.

Remarque : Si le corps du test ne comporte qu'une instruction, les accolades { } sont optionnels, (contrairement au Perl).

LES STRUCTURES DE CONTRÔLES (2)

Le test SWITCH

Le switch n'a pas d'équivalent en Perl. il est l'équivalent du *SELECT CASE* en Basic.

Il permet de confronter une variable à plusieurs valeurs prédéfinies.

Il permet un code plus compact et lisible qu'un test :

if-elseif-elseif...

Syntaxe :

```
switch( [variable] ) {
        case [valeur1] :
            [bloc d'instructions]
            break;
        case [valeur2] :
            [bloc d'instructions]
            break;
        ...
        default:
            [bloc d'instructions]
    }
```


LES STRUCTURES DE CONTRÔLES (3)

Le test SWITCH (fin)

La valeur de [variable] est comparé successivement à chaque case. Si il y a égalité, le bloc d'instruction est exécuté.

Il ne faut pas omettre le break en fin de bloc, sans quoi le reste du switch est exécuté.

Enfin, le handler default permet de définir des instructions à effectuer par défaut, c'est à dire si aucun case n'a "fonctionné"...

```
switch( $prénom ) {
case "Bob" :
case "Toto" :
case "Julien" :
echo "bonjour ", $prénom , " !  vous
     êtes un garçon";
break;

case "Anne":
case "Béatrice" :
case "Patricia" :
echo "bonjour ", $prénom , " !  vous
     êtes une fille";

default:
echo "Bonjour $prénom ! Désolé je ne
     connais pas beaucoup de
     prénoms"
}
```

LES STRUCTURES DE CONTRÔLES (4)

Les boucles

En PHP, on dispose des structures de boucle similaires au langage C.

L'instruction break permet de sortir d'une boucle à tout moment.

L'instruction continue permet de revenir au début de la boucle.

```
for( $i=0; $i < sizeof($tableau ); $i++ ) {
    if( $tableau($i] == 'suivant' ) {
    continue;
    }
    if( $tableau($i] == 'fin' ) {
    break;
    }
    echo $tableau[$i], "\n";
}
```

La boucle FOR :

```
for( [initialisations] ; [test sortie] ; [faire a chaque fois] )
    // parcours complet du tableau
    for( $i=0; $i < sizeof($tableau); $i++ ) {
    echo "tableau($i)= $tableau[$i] \n";
    }
```


LES STRUCTURES DE CONTRÔLES (5)

La boucle WHILE :

```
// parcours du tableau jusqu'au premier élément vide
$i=0;
while( isset( $tableau[$i]) ) {
echo "tableau[ $i ] = $tableau[$i] \n";
...
$i++;
}
```

La boucle DO … WHILE :

La condition de sortie est située en fin de boucle. Ainsi la boucle est parcourue une fois au minimum.

```
$fp= fopen( "monfichier.txt" );
…
do{
$ligne = fgets( $fp, 1024 );
…
} while( ! feof($fp) );
```

LES FONCTIONS (1)

A l'image de tout langage structuré, en PHP, une fonction est une suite d'instructions qui peut remplir n'importe qu'elle tâche. Tout code PHP valide figure dans le corps (ou le code) d'une fonction.

Il n'y a pas de distinction fonctions / procédures en PHP.

Les fonctions PHP prennent de 0 à n paramètres. Ces paramètres peuvent être de type quelconque.

Remarque : Il faut implémenter la fonction en amont de son utilisation, contrairement au langage C. Dans le cas contraire, PHP sort une erreur du type *Call to unsupported or undefined function (fonction) in (file) on line (number).*

On ne peut pas déclarer le prototype d'une fonction comme par exemple en Pascal.

LES FONCTIONS (2)

Déclaration :

La syntaxe de déclaration s'appuie sur le mot clé function. Ce mot clé est immédiatement suivi du nom de la fonction par lequel on va l'appeler depuis n'importe quel endroit du code PHP, puis des parenthèses destinées à accueillir les éventuels paramètres.

```
function bonjour() {
      echo " Bonjour ";
}
…..
bonjour();      // Affiche " Bonjour " à l'écran.
```

LES FONCTIONS (3)

Les fonctions peuvent ou non renvoyer un résultat. on utilise l'instruction return. La variable retournée peut être de type quelconque. Elle est transmise par copie..

```
function bonjour2() {
        return " Bonjour ";
}
…..
echo bonjour2() ;   // Affiche " Bonjour " à l'écran.
```

Le mode de fonctionnement est sensiblement différent, la fonction *bonjour* affiche directement le mot " Bonjour " à l'écran, alors que s'affiche le résultat de *bonjour2*.

LES FONCTIONS (4)

Par défaut, les variables globales ne sont pas connues à l'intérieur du corps d'une fonction. On peut cependant y accéder à l'aide du mot-clé global.

```
$debug_mode= 1; // variable globale
….
function mafonction()
{
        global $debug_mode;
        if( $debug_mode )
        echo "[DEBUG] in function mafonction()";
        ….
}
```

Une autre solution est d'utiliser le tableau associatif $GLOBALS, qui contient toutes les variables globales déclarées à un instant T : $GLOBALS['debug_mode'] équivaut à $debug_mode.

LES FONCTIONS (5)

Le passage des paramètres par valeur :

Afin de passer des paramètres à la fonction, il suffit de les insérer à l'intérieur des parenthèses prévues à cet effet.

```
function bonjour($prénom, $nom) {
    $chaîne = " Bonjour $prénom $nom " ;
    // On construit la phrase complète dans la variable locale $chaîne.
    return $chaîne ;
    // On renvoie la valeur de $chaîne comme résultat de la fonction.
}
…..
echo bonjour("Pierre" , "PAUL") ;
// Affiche " Bonjour Pierre PAUL " à l'écran.
```


LES FONCTIONS (6)

Le passage des paramètres par référence :

Par défaut, les paramètres sont transmis par copie, c'est à dire que la fonction possède une copie locale de la variable envoyée. Avec la méthode du passage des paramètres par référence, on passe à la fonction l'adresse mémoire d'une variable existante. Cela se fait en précédant de & le nom du paramètre. Cela permet de modifier ce paramètre dans la fonction.

```
function bonjour(&$phrase, $prénom, $nom) {
    $phrase = " Bonjour $prénom $nom " ;
}
…..
$chaîne = " ";
bonjour($chaîne, "Pierre" , "PAUL") ;
echo $chaîne ;          // Affiche " Bonjour Pierre PAUL " à l'écran.
```

LES FONCTIONS (7)

Le passage des paramètres par défaut :

Les paramètres optionnels sont autorisés : il suffit de leur affecter une valeur par défaut.

```
function mafonction( $param1 = "inconnu", $param2="" ) {
echo "param1=$param1 param2=$param2\n";
}
….
mafonction( "toto", "titi" ); // => "param1=toto param2=titi"
mafonction( "toto" );        // => "param1=toto param2="
mafonction();                // => "param1=inconnu param2="
```


LES FICHIERS (1)

PHP fournit plusieurs fonctions qui permettent de prendre en charge l'accès au système de fichiers du système d'exploitation du serveur.

Opérations élémentaires sur les fichiers en PHP :

- copy($source, $destination) — Copie d'un fichier,
- $fp=fopen("filemane", $mode) — Ouvre un fichier et retourne un "id" de fichier,
- fclose($fp) — Ferme un fichier ouvert,
- rename("ancien", "nouveau") — Renomme un fichier,
- fwrite($fp, $str) — Ecrit la chaîne de caractères $str,
- fputs($fp, $str) — Correspond à fwrite(),
- readfile("filename") — Lit un fichier et retourne son contenu,
- fgets($fp, $maxlength) — Lit une ligne d'un fichier,
- fread($fp, $length) — Lit un nombre donné d'octets à partir d'un fichier.

LES FICHIERS (2)

L'accès à un fichier se fait toujours par un identificateur de fichier.

Cet "id" est créé avec la fonction *fopen()* et, est requis comme paramètre par la plupart des autres fonctions de fichiers en PHP.

```
$path="/usr/local/apache/htdocs/donnees.txt";
$mode="w";
if ($fp= fopen($path, $mode) )      {
          echo "Le fichier a été ouvert";
          }
else
      echo "Fichier impossible à ouvrir";
if ( close($fp) )
      echo " et a été refermé";
?>
```


PROGRAMMATION MODULAIRES (1)

La programmation modulaire permet de la réutilisation de code, notamment par l'écriture de librairies.

De ce fait, PHP permet cette modularité par la programmation de librairies classiques et de classes.

Librairies

Les librairies sont des fichiers PHP traditionnels. Leur extension était *.inc* par convention, mais de plus en plus l'extension *.PHP* est utilisée.

On peut également inclure un fichier HTML ou d'autre type, cependant les éventuels tags PHP ne seront pas interprétés.

On inclus un fichier en utilisant les deux instructions include ou require.

PROGRAMMATION MODULAIRES (2)

Il existe une différence importante entre les deux :

➢ Un fichier inclus par *include* est inclus dynamiquement, lors de l'exécution du code, c'est-à-dire qu'il est lu puis interprété.

➢ Un fichier inclus par *require* est inclus avant l'interprétation du code. Il est équivalent à la directive *#include* du langage C.

On peut comprendre la différence sur l'exemple ci-dessous:

```
if( $user == "Administrateur" ) {
include 'admin_fonctions.php';
}
if( $user == "Administrateur" ) {
require 'admin_fonctions.php';
}
```

➔ Avec *include*, le résultat est celui escompté, tandis qu'avec *require*, le fichier *admin_fonctions.php* est inclus quelque soit le résultat du test if.

PROGRAMMATION OO (1)

Programmation Orientée Objet

PHP dispose des concepts de POO (Programmation Orientée Objet) au travers des classes.

Rappelons d'abord qu'un objet possède des attributs et des méthodes, et doit utiliser les mécanismes d'héritage et de polymorphisme.

- ✓ Attribut ➔ caractéristique d'un objet.
- ✓ Méthode ➔ action qui s'applique à un objet
- ✓ Héritage ➔ définition d'un objet comme appartenant à la même famille qu'un autre objet plus général, dont il hérite des attributs et des méthodes.
- ✓ Polymorphisme ➔ capacité d'un ensemble d'objet à exécuter des méthodes de même nom, mais dont le comportement est propre à chacune des différentes versions.

PROGRAMMATION OO (2)

Les classes

Une classe est la description complète d'un objet. Elle comprend la déclaration des attributs ainsi que l'implémentation des méthodes de cet objet.

La création d'un objet est déclenchée par celle d'une instance de la classe qui le décrit.

Une bibliothèque de composants est un ensemble de fichiers contenant des définitions de classes, que l'on peut inclure en tête des programmes qui utilisent ces classes.

Les classes peuvent être implémentées à l'aide d'autres classes. Elles sont alors définies selon le principe des couches successives, par empilage des classes de haut niveau sur des classes de bas niveau (cf. les fonctions).

PROGRAMMATION OO (3)

Déclaration

La déclaration d'une classe s'appuie sur le mot clé class. La syntaxe est comparable à celle de la déclaration des fonctions.

```
class ma_classe {
…
}
```

Affectation

Pour exploiter les méthodes et les propriétés d'un objet, on utilise un accesseur dont la syntaxe est constituée des caractères « - » et « > » côte à côte : « -> »

```
$objet_test -> ma_méthode() ;  // appelle la méthode
$objet_test -> ma_propriété ;    // accède à la propriété
```

Opérateur de la classe courante

$this-> est l'opérateur de self-référence. On peut utiliser un espace pour plus de lisibilité

- $this->nb_roues = 4 ;
- $this -> nb_roues = 4 ;

Les méthodes se déclarent comme des fonctions.

PROGRAMMATION OO (4)

Constructeur

Le constructeur se déclare comme une méthode. Il doit porter le nom de la classe comme en C++ . Il est appelé automatiquement lors de l'instanciation de la classe.

```
class Véhicule
{
var $nb_roues;

function Véhicule( $nb_roues )
{
$this-> nb_roues= $nb_roues;
}

function NbRoues()
{
return $this-> nb_roues;
}
...
}
$moto= new Véhicule( 2 );
```


PROGRAMMATION OO (5)

Héritage

L'héritage simple est possible en utilisant extends.

Remarque : le constructeur de la classe *mère* n'est pas appelé automatiquement. Il convient donc de le faire si nécessaire.

```
class Automobile extends
Véhicule
{
var $marque= "";

function Automobile( $marque,
$nb_roues )
{
$this-> Véhicule( $nb_roues );
// appel constructeur classe
parente

$this-> marque= $marque;

// set de la marque
}
}
```

PROGRAMMATION OO (6)

Limitations

Il n'y a pas notion de destructeur d'objet en PHP.

L'héritage multiple n'existe pas

Il n'y a pas de méthodes et attributs privés. Tout est public et accessible de l'extérieur.

Un objet instancié n'est pas une référence (un pointeur) mais une variable, sorte de "tableau associatif muni de méthodes". On peut s'en rendre compte sur une copie d'objet :

```
$auto= new Véhicule( 4 );
$moto= $auto;
$moto-> nb_roues= 2;
echo $auto-> nb_roues;
// 2 et non 4 => $auto et $moto sont deux objets distincts.
```


ACCES aux SGBD (1)

En général, la communication entre un programme et une base de données suit le schéma suivant :

En programmation PHP, il existe 2 méthodes pour mettre en place cette architecture :

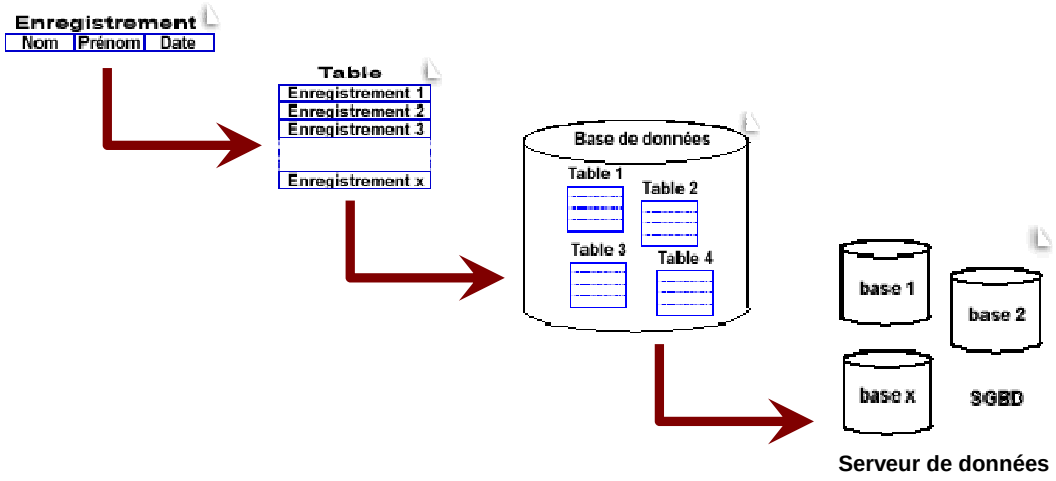
1. accéder nativement à la base par l'intermédiaire de l'API de son middleware associé,
2. passer par ODBC, l'avantage d'ODBC est de proposer une API unifiée quelque soit le SGBD utilisé.

En plus d'ODBC, PHP gère en accès natifs de nombreux SGBD :

Oracle, Sybase, Informix, MySQL, Adabas, Empress, FilePro, InterBase, mSQL, PostgreSQL, Solid, SQLServer, Unix Dbm.

ACCES aux SGBD (2)

Un SGBD est un ensemble d'applications permettant de manipuler les données (ajout, suppression, modification et lecture), mais aussi de contrôler l'accès. Les données sont structurées de la manière suivante :



ACCES aux SGBD (3)

L'utilisation en général d'un SGBD (tel que MySQL) avec PHP s'effectue en 5 temps :

1. Connexion au serveur de données
2. Sélection de la base de données
3. Requête
4. Exploitation des requêtes
5. Fermeture de la connexion

ACCES aux SGBD (4)

1- Connexion au serveur de données

Pour se connecter au serveur de données, il existe 2 méthodes :

- ✓ Ouverture d'une connexion simple avec la fonction mysql_connect
- ✓ Ouverture d'une connexion persistante avec la fonction mysql_pconnect

Remarque : la deuxième méthode diffère de la première par le fait que la connexion reste active après la fin du script.

```
<?
 if( mysql_connect("ma_base" , $login , $password ) > 0 )
  echo "Connexion réussie ! " ;
 else
  echo "Connexion impossible ! " ;
?>
```


ACCES aux SGBD (5)

2- Sélection de la base de données

Pour faire cette sélection, utilisez la fonction mysql_select_db et vous lui passez en paramètre, le nom de votre base.

```
<?
 if( mysql_select_db("ma_base" ) == True )
  echo "Sélection de la base réussie" ;
 else
  echo "Sélection de la base impossible" ;
?>
```

Remarque : les étapes sélection et requête peuvent être faites en même temps, mais il est plus simple surtout pour une seule base, de sélectionner la table avant de commencer les requêtes. Ainsi, toutes les requêtes à venir utiliseront cette base par défaut.

ACCES aux SGBD (6)

3- Envoi d'une requête

Pour envoyer ces requêtes, on peut utiliser 2 fonctions :

- ✓ mysql_query dans le cas où la base de données serait déjà sélectionnée
- ✓ mysql_db_query dans le cas où l'on voudrait sélectionner la base en même temps.

```
<?
 $requête = "SELECT * FROM membres WHERE pseudo =
'président'  ";
 $résultat = mysql_query( $requête );
?>
```


ACCES aux SGBD (7)

4- Exploitation des requêtes

Après l'exécution d'une requête de sélection, les données ne sont pas "affichées", elles sont simplement mises en mémoire, il faut les chercher, enregistrement par enregistrement, et les afficher avec un minimum de traitement.

PHP gère un pointeur de résultat, c'est celui qui est pointé qui sera retourné.
Lorsque vous utilisez une fonction de lecture, le pointeur est déplacé sur l'enregistrement suivant et ainsi de suite jusqu'à ce qu'il n'y en ait plus.

Les fonctions qui retournent un enregistrement sont : mysql_fetch_row, mysql_fetch_array et mysql_fetch_object et prennent comme paramètre l'identifiant de la requête.

Les 3 exemples suivants partent d'une requête "SELECT nom, prénom, date FROM membres."

ACCES aux SGBD (8)

mysql_fetch_row : Cette fonction retourne un enregistrement sous la forme d'un tableau simple.

```
<?
 $enregistrement = mysql_fetch_row
($résultat);

 // Affiche le champ - nom -
 echo $enregistrement[0] . "<br>";

// Affiche le champ - prénom -
 echo $enregistrement[1] . "<br> ";

// Affiche le champ - date -
 echo $enregistrement[2] . "<br> ";
?>
```

mysql_fetch_array : Cette fonction retourne un enregistrement sous la forme d'un tableau associatif.

```
<?
 $enregistrement = mysql_fetch_array
($résultat);

 // Affiche le champ - prénom -
 echo $enregistrement["prénom"]."<br>";

 // Affiche le champ - nom -
 echo $enregistrement["nom"] . "<br>";

 // Affiche le champ - date -
 echo $enregistrement["date"] . "<br>";
?>
```


ACCES aux SGBD (9)

mysql_fetch_object : Cette fonction retourne un enregistrement sous forme d'une structure (objet).

```
<?
 $enregistrement = mysql_fetch_object ($résultat );
 // Affiche le champ - date -
 echo $enregistrement->date . "<br>";
 // Affiche le champ - nom -
 echo $enregistrement->nom . "<br>";
 // Affiche le champ - prénom -
 echo $enregistrement->prénom . "<br>";
?>
```

ACCES aux SGBD (10)

Si il n'y a pas ou plus d'enregistrement à lire, ces fonctions retournent "false."

Pour savoir combien d'enregistrements ont été retournés par la sélection, la commande **mysql_num_rows** prend comme paramètre l'identifiant de la requête.

```
<?
 echo "Il y a " . mysql_num_rows( $résultat ) . " membre(s) ";
 while( $enregistrement = mysql_fetch_array( $résultat ))
 {
  echo $enregistrement['nom'] . " " . $enregistrement['prénom'];
  echo " – " . $enregistrement['date'] . "<br>" ;
 }
?>
```


ACCES aux SGBD (11)

5- Fermeture de la connexion

Vous pouvez fermer la connexion au moyen de la fonction mysql_close, mais il est bon de savoir que cette opération sera faite lorsque le script se terminera. C'est donc une opération *facultative*.

Gestion des erreurs

S'il y a une erreur dans la syntaxe de la requête, on utilise la fonction mysql_error qui ne prend pas de paramètres.

```
<?
 $résultat = mysql_query( $requête )
  or die ("Erreur dans la requête : " . $requête . "<br>Avec l\"erreur :
". mysql_error() );
?>
```

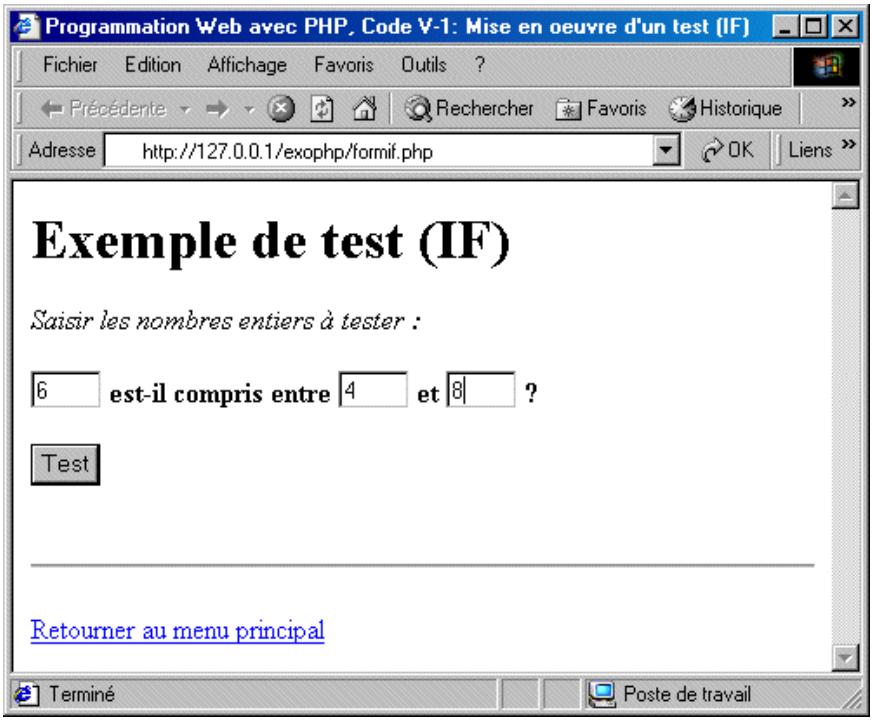
EXEMPLE (1)

Soit le programme formif.php :

```
<HTML>
<HEAD>
<TITLE>Programmation Web avec PHP, Code V-1: Mise en oeuvre d'un test (IF) </TITLE>
</HEAD>
<BODY>
<H1>Exemple de test (IF)</H1>
<i>Saisir les nombres entiers à tester :</i><br>
<FORM action="formifres.php" method=GET>
  <b><input type=text size=3 name="a"> est-il compris entre
  <input type=text size=3 name="b"> et <input type=text size=3 name="c">?</b>
  <br><br>
  <input type=submit value="Test">
</FORM>
<BR><HR><P><A href="menu.php">Retourner au menu principal</A></P>
</BODY>
</HTML>
```


EXEMPLE (2)

→ qui donne sur un navigateur cette présentation :



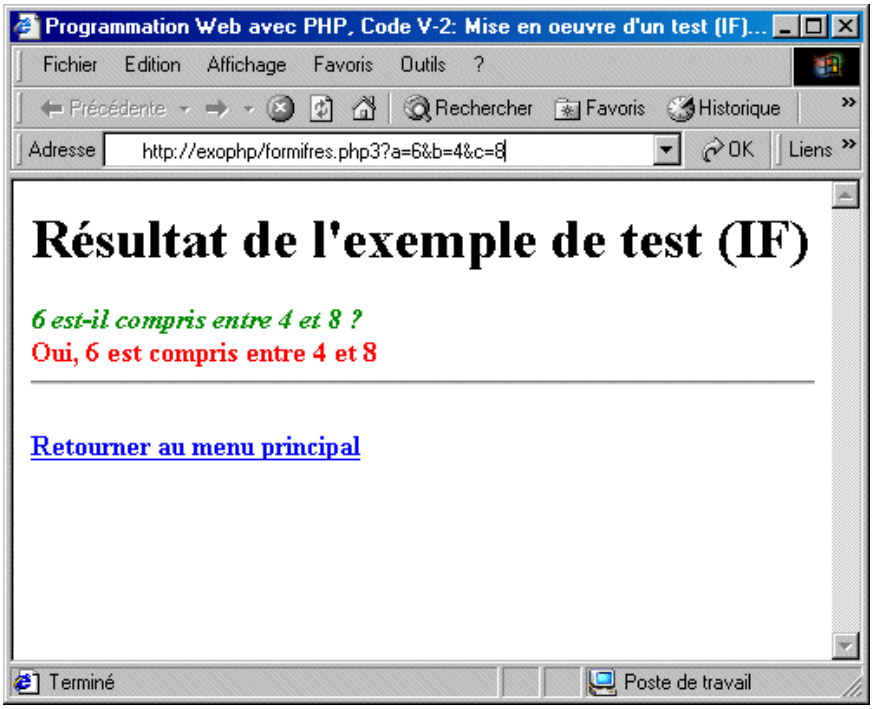
EXEMPLE (3)

Soit le programme formifres.php :

```
<HTML><HEAD><TITLE>Programmation Web avec PHP, Code V-2: Mise en
    oeuvre d'un test (IF), résultat</TITLE></HEAD>
<BODY>
<H1>Résultat de l'exemple de test (IF)</H1>
<?  $ai = intval($a);
    $bi = intval($b);
    $ci = intval($c);
    if ($ci < $bi) { $tmp = $ci;
                     $ci = $bi;
                     $bi = $tmp; }
    echo "<font color=\"008800\"><b><i>$ai est-il compris entre $bi et $ci
    ?<br></i>";
    echo "</font><font color=\"ff0000\">";
    if ($ai < $bi) {echo "Non, $ai est inférieur à $bi " ;
                    } elseif ($ai > $ci) { echo "Non, $ai est supérieur à $ci " ;
                    } else { echo "Oui, $ai est compris entre $bi et $ci " ; }
?>
<BR><HR><P><A href="menu.php">Retourner au menu principal</A></P>
</BODY></HTML>
```


EXEMPLE (4)

→ Ce qui donne sur un navigateur :



PHP ←→ ASP (1)

Quelques comparaisons entre PHP et ASP

- ✓ PHP (4 seulement) possède l'équivalent des Sessions ASP. Cependant, il existe des librairies (pour PHP3) qui implémente la Session.
- ✓ L'éventail de fonctions PHP est nettement supérieur. (plus de 500 fonctions).
- ✓ PHP implémente la programmation orientée objet.
- ✓ PHP reconnaît les balises ASP <% et %> ainsi que <%= (permettant l'affichage de variables). Il suffit pour cela de modifier la configuration: asp_tags = On.
- ✓ PHP gère en standard -et simplement- le *File Upload.*
- ✓ PHP implémente les expressions régulières.
- ✓ En plus d'ODBC, PHP gère en accès natifs de nombreux SGBD:
  Oracle, Sybase, Informix, MySQL, Adabas, Empress, FilePro, InterBase, mSQL, PostgreSQL, Solid, SQLServer, Unix Dbm.

PHP ←→ ASP (2)

PHP	ASP
<? code; ?>	<% code %>
syntaxe JavaScript	syntaxe VBScript
<? // commentaire ?>	<% ' commentaire %>
$variable	variable
<? echo "con" . "caténation"; ?>	<% ="con" & "caténation" %>
<? Header("Location: page.htm"); ?>	<% response.redirect "page.htm" %>
conditionnelle <? include("truc.php"); ?> *brute* <? require "truc.php"; ?>	*relative* <!-- #include file="truc.asp" --> *absolue* <!-- #include virtual="/truc.asp" -->
<? if($myvar) { ?> ... <? } else { ?> ... <? } ?>	<% if myvar <> "" then %> ... <% else %> ... <% end if %>

PHP ←→ ASP (3)

PHP	ASP
(méthode POST) <? echo $champ1; ?>	*(méthode POST)* <% =request.form("champ1") %>
(méthode GET) <? echo $langue; ?>	*(méthode GET)* <% =request.querystring("langue") %>
<? setcookie("asphp", "toto",time()+86400); ?> <? echo $asphp; ?> <? setcookie("asphp"); ?>	*Ecrire* <% response.cookies("asphp")="toto" response.cookies("asphp").Expires = Date+1 %> *Lire* <% =request.cookies("asphp") %> *Détruire* <% response.cookies("asphp")="" %>
<? $bool=mail(...); ?>	<% bool=Mail.sendMail %>
<? echo getenv(...); ?>	<% =request.servervariables(...) %>
PHP4 <? session_register("email"); $email="info@asp-php.net"; echo $email; ?>	<% Session("email")="info@asp-php.net" %> <% =Session("email") %>

PHP ←→ ASP (4)

PHP	ASP
SGBD MySQL : <? mysql_connect($host,$user,$pass); mysql_select_db("$bdd"); ?>	SGBD Access : <% Set Conn = Server.CreateObject("ADODB.Connection") Conn.Open DSN %>
<? $inF = fopen($Fnm,"w"); fputs($inF,"Bonjour"); fclose($inF); ?>	<% set inF = FSO.CreateTextFile(Fnm) inF.WriteLine("Bonjour") inF.Close %>
<? $inF = fopen($Fnm,"r"); echo fgets($inF, 4096); fclose($inF); ?>	<% set inF = FSO.OpenTextFile(Fnm,1,false) %> <% =inF.ReadLine %> <% inF.Close %>

PHP ←→ AJAX (1)

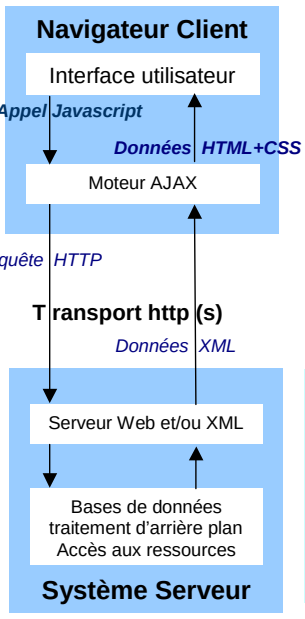
AJAX : le concept

AJAX (pour Asynchronous JavaScript and XML) n'est pas un nouveau langage de programmation, mais plutôt une combinaison de différentes techniques déjà existantes. En règle général, on marie les ingrédients suivants:

- ✓(X)HTML et CSS pour la description du contenu
- ✓JavaScript pour l'interaction du côté du client
- ✓PHP (ou un autre langage de programmation) pour l'interaction du côté du serveur
- ✓XML (ou un autre format d'échange de données) pour le contenu
- ✓L'objet XMLHttpRequest (avec Firefox, Safari, Opera) ou Microsoft.XMLHTTP (sous Internet Explorer) pour effectuer des transferts en tâche de fond

Principe : une première page s'affiche, puis différents événements détectés en JavaScript lancent ensuite des requêtes du côté du serveur.

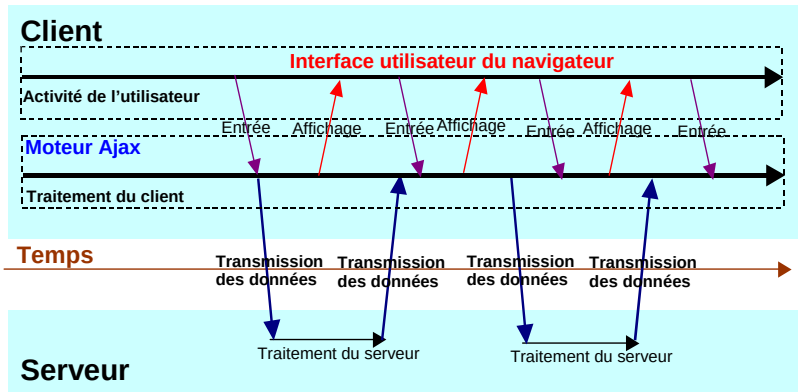
PHP ←→ AJAX (2)



Modèle d'application Web Ajax

AJAX n'est donc pas un nouveau langage de programmation, mais plutôt un concept qui permet de faire des appels asynchrones au serveur depuis le client. Pour cela on utilise le langage JavaScript, et puis XML avec l'objet XMLHttpRequest pour les transmettre de façon asynchrone sur le poste serveur.

Lors de ces appels, le serveur retournera des données formatées – souvent en XML- qui seront récupérées et traitées à l'aide d'un programme JavaScript.



PHP ←→ AJAX (3)

Principe d'interaction entre PHP et AJAX

- Etape 1 : il faut avoir stocké les données dans des fichiers en format xml. Ces fichiers peuvent être soit le résultat d'une requête SQL (*select * from table)* ou un fichier comportant des données, des paramètres… respectant une certain format.
  ➔ Ce fichier sera appelé et mis à jour de manière asynchrone à la demande du client.
- Etape 2 : il faut ensuite créer un objet de la classe XMLHttpRequest pour récupérer le contenu du fichier XML qui sera traité et affiché en utilisant une fonction JavaScript (complétée éventuellement par une feuille de style CSS).
  - ✓ création d'une instance de XMLHttpRequest grâce à l'appel de fonction new XMLHttpRequest()
  - ✓ association de l'arbre XML à ce nouvel objet avec une méthode d'envoi « GET » ou « POST », il suffit pour se faire d'appeler la fonction *open()* de l'objet avec comme paramètre l'adresse où se trouve le fichier XML sur le serveur.
- Etape 3 : il ne reste plus qu'à traiter ce fichier car à ce stade on a la réponse du serveur en arrière plan, c'est-à-dire sans que la page affichée sur le navigateur ne bouge, on pourra donc l'interroger avec du code JavaScript et puis mettre à jour certaines parties de la page (avec le même langage)

PHP ←→ AJAX (XMLHttpRequest)


```
XMLHttpRequest xmlHttp = new XMLHttpRequest();
// Corps de la fonction
function callingPage (idDiv, rept, cible, extension)
{// ne continuer que si l'objet xmlHttp n'est pas vide (instancié)
  if (xmlHttp)
  { // Tentative de connexion au serveur
    try {
//valeurs (éventuelles) saisies par l'utilisateur, pas de valeur dans ce cas
// construction du chemin d'accès au fichier selon les paramètres de la fonction
      var chemin = rept+cible.text+"."+extension; // le répertoire automatiquement
généré
      var POSTparams = "";
// L'appel asynchrone (true), avec la méthode POST, vers la variable chemin
        xmlHttp.open("POST", chemin, true);
        xmlHttp.onreadystatechange = function(){
// Récupérer Lors que readyState de l'objet xmlHttp a la valeur 4 et son status=200
  if (xmlHttp.readyState == 4){
        if (xmlHttp.status == 200){
        try {
//faire quelque chose de la réponse du serveur : xhr.responseXXX ....
            document.title=cible.text;
//titre de la page est mis à jour, c'est le nom du fichier
            var reponse = xmlHttp.responseText;
```

PHP ←→ AJAX (XMLHttpRequest)


```
// Le contenu de la div passée en paramètre à la valeur reponse, envoyé du serveur
            document.getElementById(idDiv).innerHTML=reponse;
            }
        catch(e) {
            alert("Error reading the response : " + e.toString());}
            }
            else {// Afficher un message si le fichier n'existe pas
                alert("Un problème a été rencontré en chargeant le données ***:\n"
+ xmlHttp.statusText);
                }
              }
            }
//cette ligne est ajoutée si et seulement si la méthode de open est POST
            xmlHttp.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
            xmlHttp.send(POSTparams);
    }
//afficher un message d'erreur si l'objet xllHttp=null
    catch (e)
    {
     alert("Connexion au serveur impossible :\n" + e.toString());
    }
  }
}//fin de la fonction
```


PHP ←→ AJAX (6)

Exemple d'invocation d'une méthode à partir d'un lien :

- La méthode s'appelle **Ma_méthode()** et possède 4 paramètres (l'identifiant de la div à mettre à jour, le répertoire où se trouve le fichier XML, le nom du fichier et son extension), elle tient compte des déclenchements d'exceptions grâce aux blocs TRY… CATCH.

- La variable **xmlHttp** représente l'objet **XMLHttpRequest** qui est instancié à l'extérieur de la fonction dans le fichier JavaScript.

```
<a href="javascript:Ma_méthode( 'CORP' , 'AJAX/RESSOURCES/' , 'Ressources1' , 'php' , 'Ressources' );"> <img src = "Images/gestion_ressources.PNG" height="90" class="imgMenue" /></a>
```

BIBLIOGRAPHIE

Les sites Web :

- **http://www.php.net/ (Site officiel PHP)**
- **http://www.phpindex.com/**
- **http://www.phpfrance.com/**
- **http://www.phpinfo.net/**
- **http://www.phpdebutant.com/**
- **http://www.ilovephp.com/**
- **http://www.asp-php.com/**
- **http://www.mysql.org/ (Site officiel MYSQL)**

Mais aussi les ouvrages :

- **Programmation Web avec PHP – C. Lacroix, N. Leprince, C. Boggero, C. Lauer – éditions Eyrolles**
- **Vos premiers pas avec PHP 4 – J. Engels – éditions Eyrolles**
- **Grand livre PHP4 & MySQL – G. Leierer, R. Stoll – éditions Eyrolles**