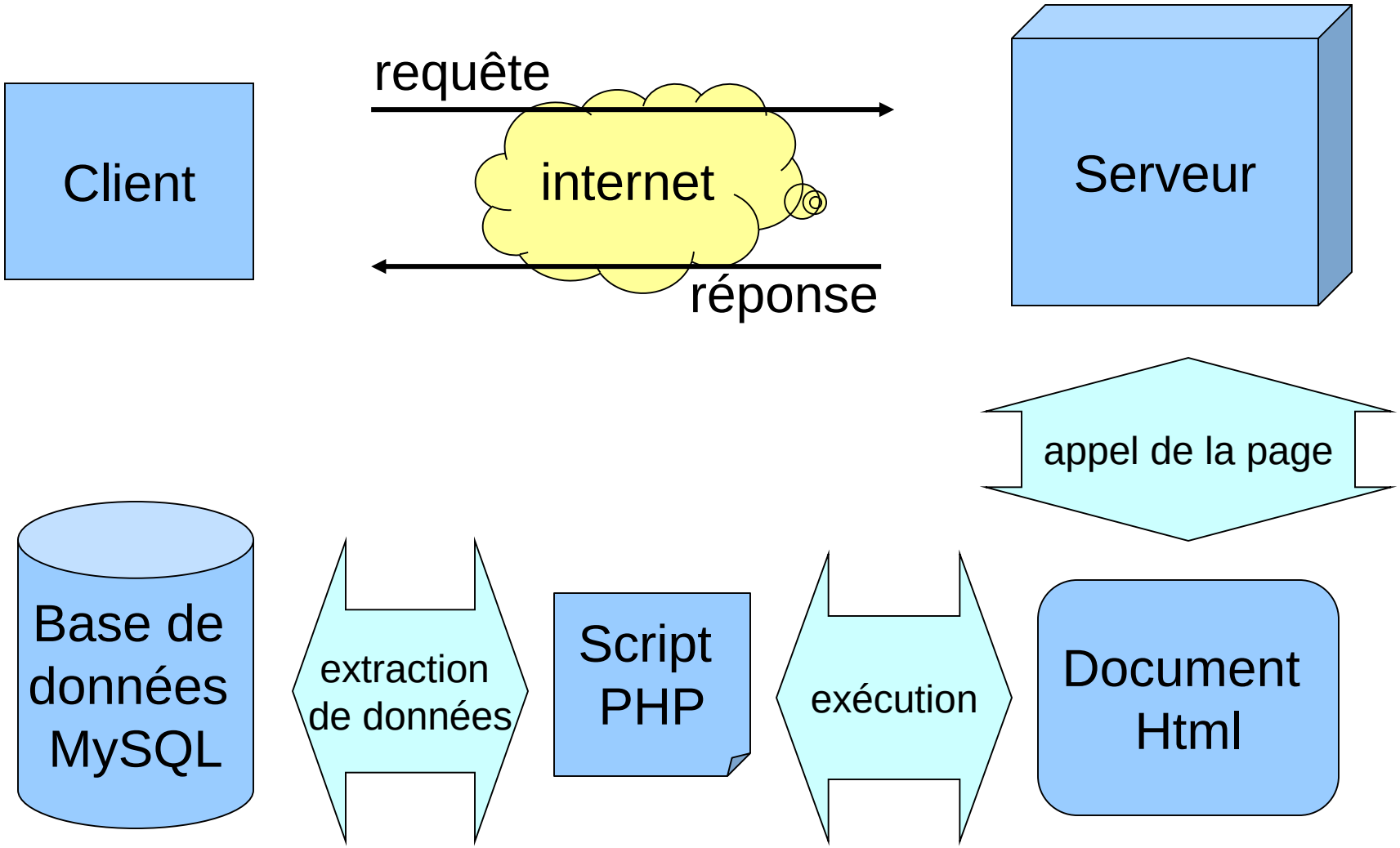
Programmation web et Multimédia

PHP4 et MySQL

2013/2014

Principe


Client
requête
internet
réponse
Serveur
appel de la page
Base de données MySQL
extraction de données
Script PHP
exécution
Document Html

Environnement

Pour essayer les exemples du cours, il faut installer un serveur web *Apache* avec *PHP* et *MySQL*

- Wamp (http://www.wampserver.com/)
- Easyphp (http://www.easyphp.org/)
- Etc

N’oubliez pas de configurer le serveur.

Pour essayer un exemple, créer un fichier page.php sous le répertoire www de votre serveur

Dans le navigateur tapez l’url du fichier à invoquer

http://localhost:NN/page.php

Intégration d'un script dans une page

Les pages web sont au format html. Les pages web dynamiques générées avec PHP sont au format php. Le code source php est directement inséré dans le fichier html grâce au conteneur de la norme XML :

<?php … ?>

Exemple:

```
<html>
<body>
<?php
        echo ''Bonjour'';
?>
</body>
</html>
```

Autres syntaxes d'intégration :

- **<? … ?>**
- **<script language=''php''> … </script>**

Exemple de script

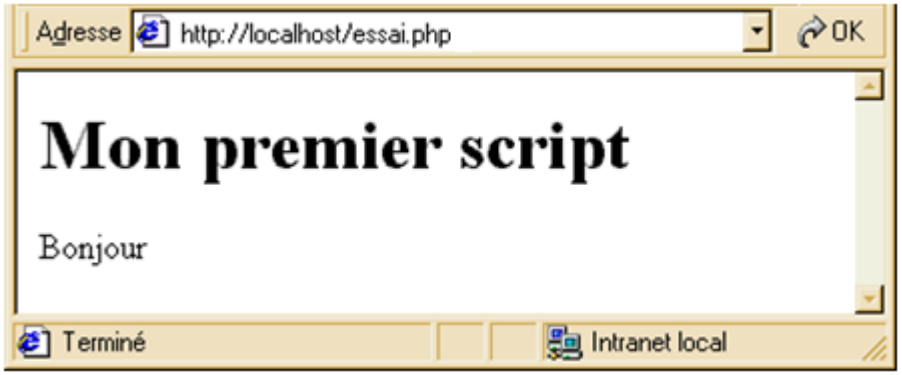
Exemple de script, code source (côté serveur) :

```
<html>
<body>
<h1>Mon premier script</h1>
<?php echo ‘’Bonjour\n”; ?>
</body>
</html>
```

Autre écriture du même script :

```
<?php
echo ‘’<html>\n<body>\n”;
echo ‘’<h1>Mon premier script</h1>\n”;
echo ‘’Bonjour\n”;
echo ‘’</body>\n</html>\n”;
?>
```

Résultat affiché par le navigateur :



Commentaires

Un script php se commente comme en langage C :

Exemple :

```
<?php

// commentaire de fin de ligne

/* commentaire
sur plusieurs
lignes */

?>
```

Remarque

Tout ce qui se trouve dans un commentaire est ignoré. Il est conseillé de commenter largement ses scripts.

Variables, types et opérateurs (I)

Le typage des variables est implicite en php. Il n'est donc pas nécessaire de déclarer leur type au préalable ni même de les initialiser avant leur utilisation.

Les identificateurs de variable sont précédés du symbole « **$** » (dollars). Exemple : **$toto**.

Les variables peuvent être de type entier (**integer**), réel (**double**), chaîne de caractères (**string**), tableau (**array**), objet (**object**), booléen (**boolean**).

Il est possible de convertir une variable en un type primitif grâce au cast (comme en C).

Exemple :

```
$str = ''12'';                  // $str vaut la chaîne ''12''
$nbr = (int)$str;               // $nbr vaut le nombre 12
```

Variables, types et opérateurs (II)

Quelques fonctions :

empty($var) : renvoie vrai si la variable est vide

isset($var) : renvoie vrai si la variable existe

unset($var) : détruit une variable

gettype($var) : retourne le type de la variable

settype($var, ''type'') : convertit la variable en type **type** (cast)

is_long(), **is_double()**, **is_string()**, **is_array()**, **is_object()**, **is_bool()**, **is_float()**, **is_numeric()**, **is_integer()**, **is_int()…**

Une variable peut avoir pour identificateur la valeur d'une autre variable.

Syntaxe : **${$var} = valeur;**

Exemple :

```
$toto = ''chaine'';
${$toto} = 2012;
echo $chaine;                    // la variable $chaine vaut 2012
```

Variables, types et opérateurs (III)

La portée d'une variable est limitée au bloc dans lequel elle a été crée. Une variable locale à une fonction n'est pas connue dans le reste du programme. Tout comme une variable du programme n'est pas connue dans une fonction. Une variable créée dans un bloc n'est pas connue dans les autres blocs, mêmes supérieurs.

Opérateurs arithmétiques :

+ (addition), **-** (soustraction), ***** (multiplié), ***/*** (divisé), **%** (modulo), **++** (incrément), **--**(décrément). Ces deux derniers peuvent être pré ou post fixés

Opérateurs d'assignement :

= (affectation), ***=** (**$x*=$y** équivalent à **$x=$x*$y**), ***/=***, **+=**, **-=**, **%=**

Opérateurs logiques :

and, **&&** (et), **or**, **||** (ou), **xor** (ou exclusif), **!** (non)

Opérateurs de comparaison :

== (égalité), **<** (inférieur strict), **<=** (inférieur large), **>**, **>=**, **!=** (différence)

Variables, types et opérateurs (IV)

Signalons un opérateur très spécial qui équivaut à une structure conditionnelle complexe *if then else* à la différence qu'il renvoie un résultat de valeur pouvant ne pas être un booléen : **l'opérateur ternaire**.

Syntaxe :

```
(condition)?(expression1):(expression2);
```

Si la **condition** est vrai alors évalue et renvoie l'**expression1** sinon évalue et renvoie l'**expression2**.

Exemple :

```
$nbr = ($toto>10)?($toto):($toto%10);
```

Dans cet exemple, la variable **$nbr** prend **$toto** pour valeur si **$toto** est strictement supérieur à **10**, sinon vaut le reste de la division entière de **$toto** par **10**.

Constantes

L'utilisateur peut définir des constantes dont la valeur est fixée une fois pour toute. Les constantes ne portent pas le symbole **$** (dollars) en début d'identificateur et ne sont pas modifiables.

define(''var'',valeur) : définit la constante **var** (sans **$**) de valeur **valeur**

Exemple 1 :

```
define(''author'',''auteur'');
echo author;                    // affiche '' auteur
```

Exemple 2 :

```
define(MY_YEAR,1984);
echo MY_YEAR;                   // affiche 1984
```

Contrairement aux variables, les identificateurs de constantes (et aussi ceux de fonction) ne sont pas sensibles à la casse.

Références

On peut à la manière des pointeurs en C faire référence à une variable grâce à l'opérateur **&** (ET commercial).

Exemple 1 :

```
$toto = 100;          // la variable $toto est initialisée à la valeur 100
$var= &$toto; // la variable $var fait référence à $toto
$toto++;              // on change la valeur de $toto
echo $var;            // qui est répercutée sur $var qui vaut alors 101
```

Exemple 2 :

```
function change($var) {
        $var++;   // la fonction incrémente en local l'argument
}
$nbr = 1;             // la variable $nbr est initialisée à 1
change(&$nbr);    // passage de la variable par référence
echo $nbr;            // sa valeur a donc été modifiée $nbr=2
```

Mathématiques

Il existe une myriade de fonctions mathématiques.

abs($x) : valeur absolue

ceil($x) : arrondi supérieur

floor($x) : arrondi inférieur

pow($x,$y) : x exposant y

max($a, $b, $c …) : retourne l'argument de valeur maximum

pi() : retourne la valeur de Pi

Et aussi : **cos**, **sin**, **tan**, **exp**, **log**, **min**, **pi**, **sqrt**…

Quelques constantes :

valeur de pi (3.14159265358979323846)

valeur de e (2.71828182845904523 54)

Nombres aléatoires :

rand([$x[,$y]) : valeur entière aléatoire entre x et y .

Booléens

Les variables booléennes prennent pour valeurs **TRUE** (vrai) et **FALSE** (faux). Une valeur entière nulle est automatiquement considérée comme **FALSE**. Tout comme une chaîne de caractères vide **‘’’**. Ou encore comme les chaînes **‘’0’’** et **‘0’.**

Exemple :

```
if(0) echo 1;        // faux
if("") echo 2;       // faux
if("0") echo 3;      // faux
if("00") echo 4;
if('0') echo 5;      // faux
if('00') echo 6;
if(" ") echo 7;
```

Cet exemple affiche **467**. Donc l’espace ou la chaîne **‘’00’’** ne sont pas considérés **FALSE**.

Chaînes de caractères (I)

Une variable chaîne de caractères n'est pas limitée en nombre de caractères. Elle est toujours délimitée par des simples quotes ou des doubles quotes.

Exemples :

$prenom = ‘’Victor’’;

$nom = ‘Hugo’;

Les doubles quottes permettent l'évaluation des variables et caractères spéciaux contenus dans la chaîne (comme en C ou en Shell) alors que les simples ne le permettent pas.

Exemples :

echo ‘’Nom: $nom‘’; // affiche **Nom: Hugo**

echo ’Nom: $nom‘; // affiche **Nom: $nom**

Quelques caractères spéciaux : **\n** (nouvelle ligne), **\r** (retour à la ligne), **\t** (tabulation horizontale), **** (antislash), **\$** (caractère dollars), **\’’** (double quote).

Exemple : **echo ‘’Hello Word !\n’’;**

Chaînes de caractères (II)

Opérateur de concaténation de chaînes : **.** (point)

Exemple 1 :

```
$var1 = ‘’Hello’’;
$var2 = ‘’Word’’;
echo $var1.$var2;
```

Exemple 2 :

```
$name = ‘’Henry’’;
$whoiam = $name.‘’IV’’;
```

Exemple 3 :

```
$out = ‘Patati’;
$out .= ‘’ et patata…’’;
```

Chaînes de caractères (III)

Affichage d'une chaîne avec **echo**:

Exemples:

echo 'Hello Word.';

echo ''Hello $name\n'';

echo ''Nom : '', $name;

echo(''Bonjour'');

Quelques fonctions:

strlen($str) : retourne le nombre de caractères d'une chaîne

strtolower($str) : conversion en minuscules

strtoupper($str) : conversion en majuscules

trim($str) : suppression des espaces de début et de fin de chaîne

substr($str,$i,$j) : retourne une sous chaîne de **$str** de taille **$j** et débutant à la position **$i**

strnatcmp($str1,$str2) : comparaison de 2 chaînes

ord($char) : retourne la valeur ASCII du caractère **$char**

Structures de contrôle (I)

Structures conditionnelles (même syntaxe qu'en langage C) :

```
if( ... ) {
        …
} elseif {
        …
} else {
        …
}

switch( … ) {
        case … : { … } break
        …
        default : { … }
}
```

Structures de contrôle (II)

Structures de boucle (même syntaxe qu'en langage C) :

```
for( … ; … ; … ) {
        …
}

while( … ) {
        …
}

do {
        …
} while( … );
```

Structures de contrôle (III)

L'instruction **break** permet de quitter prématurément une boucle.

Exemple :

```
while($nbr = $tab[$i++]) {
        echo $nbr.''<br />'';
        if($nbr == $stop)
                break;
}
```

L'instruction **continue** permet d'éviter les instructions suivantes de l'itération courante de la boucle pour passer à la suivante.

Exemple :

```
for($i=1; $i<=10; $i++) {
        if($tab[$i] == $val)
                continue;
        echo $tab[$i];
}
```

Tableaux (I)

Une variable tableau est de type **array**. Un tableau accepte des éléments de tout type. Les éléments d'un tableau peuvent être de types différents et sont séparés d'une virgule.

Un tableau peut être initialisé avec la syntaxe **array**.

Exemple :

```
$tab_colors = array('red', 'yellow', 'blue', 'white');
$tab = array('foot', 2013, 20.5, $name);
```

Mais il peut aussi être initialisé au fur et à mesure.

Exemples :

```
$prenoms[ ] = ''Clément'';        $villes[0] = ''Paris'';
$prenoms[ ] = ''Justin'';         $villes[1] = ''Londres'';
$prenoms[ ] = ''Tanguy'';         $villes[2] = ''Lisbonne'';
```

L'appel d'un élément du tableau se fait à partir de son indice (dont l'origine est zéro comme en C**)**.

Exemple : **echo $tab[10];** // pour accéder au 11ème élément

Tableaux (II)

Parcours d'un tableau.

$tab = array('Hugo', 'Jean', 'Mario');

Exemple 1 :

```
$i=0;
while($i <= count($tab)) {        // count() retourne le nombre d'éléments
  echo $tab[$i].''\n'';
  $i++;
}
```

Exemple 2 :

```
foreach($tab as $elem) {
  echo $elem.''\n'';
}
```

La variable **$elem** prend pour valeurs successives tous les éléments du tableau **$tab**.

Tableaux (III)

Quelques fonctions:

count($tab), **sizeof** : retournent le nombre d'éléments du tableau

in_array($var,$tab) : dit si la valeur de **$var** existe dans le tableau **$tab**

list($var1,$var2…) : transforme une liste de variables en tableau

shuffle($tab) : mélange les éléments d'un tableau

sort($tab) : trie alphanumérique les éléments du tableau

rsort($tab) : trie alphanumérique inverse les éléments du tableau

implode($str,$tab), join : retournent une chaîne de caractères contenant les éléments du tableau **$tab** joints par la chaîne de jointure **$str**

explode($delim,$str) : retourne un tableau dont les éléments résultent du hachage de la chaîne **$str** par le délimiteur **$delim**

array_merge($tab1,$tab2,$tab3…) : concatène les tableaux passés en arguments

array_rand($tab) : retourne un élément du tableau au hasard

Tableaux associatifs (I)

Un tableau associatif est appelé aussi *dictionnaire*. On associe à chacun de ses éléments une clé dont la valeur est de type chaîne de caractères.

L'initialisation d'un tableau associatif est similaire à celle d'un tableau normal.

Exemple 1 :

$personne = array(''Nom'' => ''César'', ''Prénom'' => ''Jules'');

Exemple 2 :

$personne[''Nom''] = ''César'';

$personne[''Prénom''] = ''Jules'';

Ici à la clé **''Nom''** est associée la valeur **''César''**.

Tableaux associatifs (II)

Parcours d'un tableau associatif.

$personne = array(''Nom'' => ''César'', ''Prénom'' => ''Jules'');

Exemple 1 :

```
foreach($personne as $elem) {
  echo $elem;
}
```

Ici on accède directement aux éléments du tableau sans passer par les clés.

Exemple 2 :

```
foreach($personne as $key => $elem) {
  echo ''$key : $elem'';
}
```

Ici on accède simultanément aux clés et aux éléments.

Tableaux associatifs (III)

Quelques fonctions :

array_count_values($tab) : retourne un tableau contenant les valeurs du tableau **$tab** comme clés et leurs fréquence comme valeur (utile pour évaluer les redondances)

array_keys($tab) : retourne un tableau contenant les clés du tableau associatif **$tab**

array_values($tab) : retourne un tableau contenant les valeurs du tableau associatif **$tab**

array_search($val,$tab) : retourne la clé associée à la valeur **$val**

L'élément d'un tableau peut être un autre tableau.

Tableaux associatifs (IV)

Quelques fonctions alternatives pour le parcours de tableaux (normaux ou associatifs) :

reset($tab) : place le pointeur sur le premier élément

current($tab) : retourne la valeur de l'élément courant

next($tab) : place le pointeur sur l'élément suivant

prev($tab) : place le pointeur sur l'élément précédant

each($tab) : retourne la paire clé/valeur courante et avance le pointeur

Exemple :

```
$colors = array(''red'', ''green'', ''blue'');
$nbr = count($colors);
reset($colors);
for($i=1; $i<=$nbr; $i++) {
        echo current($colors).''<br />'';
        next($colors);
}
/* il est aussi possible d'utiliser la  structure foreach dans ce cas*/
```

Fonctions (I)

Comme tout langage de programmation, php permet l'écriture de fonctions.

Les fonctions peuvent prendre des arguments dont il n'est pas besoin de spécifier le type. Elles peuvent de façon optionnelle retourner une valeur.

L'appel à une fonction peut ne pas respecter son prototypage (nombre de paramètres). Les identificateurs de fonctions sont insensibles à la casse.

Exemple :

```
function mafonction($toto) {
        $toto += 15;
        echo ''Salut !'';
        return ($toto+10);
}
$nbr = MaFonction(15.1);
/* retourne 15.1+15+10=40.1, les majuscules n'ont pas d'importance */
```

Fonctions (II)

On peut modifier la portée des variables locales à une fonction.

global permet de travailler sur une variable de portée globale au programme. Le tableau associatif **$GLOBALS** permet d'accéder aux variables globales du script (**$GLOBALS[''var'']** accède à la variable **$var**).

Exemple :

```
function change() {
        global $var;       // définit $var comme globale
        $GLOBALS[''toto''] ++; // incrémente la variable globale $toto
        $var++;            // cela sera répercuté dans le reste du programme
}
```

static permet de conserver la valeur d'une variable locale à une fonction.

Exemple :

```
function change() {
        static $var;      // définit $var comme statique
        $var++;           // sa valeur sera conservée jusqu'au prochain appel
}
```

Inclusions

On peut inclure dans un script php le contenu d'un autre fichier.

require insert dans le code le contenu du fichier spécifié même si ce n'est pas du code php. Est équivalent au préprocesseur *#include* du C.

Exemple :

require(''fichier.php'');

include évalue et insert à chaque appel (même dans une boucle) le contenu du fichier passé en argument.

Exemple :

include(''fichier.php'');

PHP et les formulaires (I)

Pour la récupération des paramètres en PHP, on utilise les tableaux associatifs **$_GET** et **$_POST** qui contiennent toutes les variables envoyées par un formulaire

```
<form method="GET | POST" action="page_cible.php">
<input type="text" name="Champ_saisie" value="Texte« >
<select name="Liste_Choix" size="3">
<option value="Option_1">Option_1</option>
<option value="Option_2">Option_2</option>
<option value="Option_3">Option_3</option>
</select>
<textarea name="Zone_Texte" cols="30" rows="5"> Texte par défaut
</textarea>
   <input type="checkbox" name="Case_Cocher[]" value="Case_1"> Case 1<br>
   <input type="checkbox" name="Case_Cocher[]" value="Case_2"> Case 2<br>
   <input type="checkbox" name="Case_Cocher[]" value="Case_3"> Case 3<br>
   <input type="radio" name="Case_Radio" value="Case radio 1"> radio
   1<br>
   <input type="radio" name="Case_Radio" value="Case radio 2"> radio
   2<br>
   <input type="radio" name="Case_Radio" value="Case radio 3"> radio
   3<br>
   <input type="reset" name="Annulation" value="Annuler">
 <input type="submit" name="Soumission" value="Soumettre">
</form>
```

PHP et les formulaires (II)

Code de récupération des données.

```
<?php
  $resultat = $_GET["Champ_saisie"] ; ?>
 . "<br>";
  $resultat .= $_GET["Liste_Choix"] . "<br>";
  $resultat .= $_GET["Zone_Texte"] . "<br>";
  for ($i = 0; $i < count($_GET["Case_Cocher"]); $i++)
  {
    $resultat .= $_GET["Case_Cocher"][$i] . "<br>";
  }
  $resultat .= $_GET["Case_Radio"] . "<br>";
  echo $resultat;
  ?>
```

Interfaçage avec la base de données

PHP propose de nombreux outils permettant de travailler avec la plupart des SGBD :Oracle, Sybase, Microsoft SQL Server, PostgreSQL ou encore MySQL.

Avec le SGBD MySQL, les fonctions principales de manipulation sont :

mysql_connect : La fonction de connexion au serveur

mysql_select_db : La fonction de choix de la base de données

mysql_query : La fonction de requête

mysql_close : La fonction de déconnexion

Connexion (I)

Pour se connecter à une base depuis php, il faut spécifier un nom de serveur, un nom d'utilisateur, un mot de passe et un nom de base.

Les fonctions de connexion :

mysql_connect($server,$user,$password) : permet de se connecter au serveur **$server** en tant qu'utilisateur **$user** avec le mot de passe **$password**, retourne l'identifiant de connexion si succès, FALSE sinon

mysql_select_db($base[,$id]) : permet de choisir la base **$base**, retourne TRUE en cas de succès, sinon FALSE

mysql_close([$id]) : permet de fermer la connexion

mysql_pconnect : idem que **mysql_connect** sauf que la connection est persistante, il n'y a donc pas besoin de rouvrir la connexion à chaque script qui travaille sur la même base.

Les identifiants de connexion ne sont pas nécessaires si on ne se connecte qu'à une seule base à la fois, ils permettent seulement de lever toute ambiguïté en cas de connexions multiples.

Connexion (II)

Exemple 1 :

```
if( $id = mysql_connect(‘’localhost’’,’’utilisateur’’,’’0478’’) ) {
        if( $id_db = mysql_select_db(‘’Mabase’’) ) {
                echo ‘’Succès de connexion.’’;
                /* code du script … */
        } else {
                die(‘’Echec de connexion à la base.’’);
        }
        mysql_close($id);
} else {
        die(‘’Echec de connexion au serveur de base de données.’’);
}
```

La fonction **die** arrête un script et affiche un message d’erreur dans le navigateur.

Connexion (III)

Exemple 2 :

```
@mysql_connect(‘’localhost’’,’ ’utilisateur’’,’’0478’’) or die(‘’Echec de connexion au serveur.’’);
@mysql_select_db(‘’Mabase’’) or die(‘’Echec de sélection de la base.’’);
```

Cet exemple est équivalent au précédent mais plus court à écrire. Le symbole @ (arobase) permet d’éviter le renvoie de valeur par la fonction qu’il précède.

On pourra avantageusement intégrer ce code dans un fichier que l’on pourra joindre par **include()**. C’est aussi un moyen de sécuriser le mot de passe de connexion.

Une connexion persistante évite d’avoir à rouvrir une connexion dans chaque script. Les connexions sont automatiquement fermées au bout d’un certain temps en cas d’absence de toute activité…

Interrogation

Pour envoyer une requête à une base de donnée, il existe la fonction : **mysql_query($str)** qui prend pour paramètre une chaîne de caractères qui contient la requête écrite en SQL (Structured Query Language) et retourne un identificateur de résultat ou FALSE si échec.

Les requêtes les plus couramment utilisées sont : **CREATE** (création d'une table), **SELECT** (sélection), **INSERT** (insertion), **UPDATE** (mise à jour des données), **DELETE** (suppression), **ALTER** (modification d'une table), etc.

Exemple :

$result = mysql_query(‘’SELECT address FROM users WHERE name=\’’$name\’’’’);

Cet exemple recherche l'adresse de l'utilisateur portant pour nom la valeur de la chaîne **$name**. L'identificateur de résultat **$result** permettra à d'autres fonctions d'extraire ligne par ligne les données retournées par le serveur.

Extraction des données (I)

Une fois la requête effectuée et l'identificateur de résultat acquis, il ne reste plus qu'à extraire les données retournées par le serveur

mysql_fetch_row($result) : retourne une ligne de résultat sous la forme d'un tableau. Les éléments du tableau étant les valeurs des attributs de la ligne. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 1 :

```
$requete = ''SELECT * FROM users'';
if($result = mysql_query($requete)) {
        while($ligne = mysql_fetch_row($result)) {
                $id = $ligne[0];
                $name = $ligne[1];
                $address = $ligne[2];
                echo ''$id - $name, $address <br />'';
        }
} else {
        echo ''Erreur de requête de base de données.'';
}
```

Ici, on accède aux valeurs de la ligne par leur indice dans le tableau.

Extraction des données (II)

mysql_fetch_array($result) : retourne un tableau associatif. Les clés étant les noms des attributs et leurs valeurs associées leurs valeurs respectives. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 2 :

```
$requete = ''SELECT * FROM users'';
if($result = mysql_query($requete)) {
        while($ligne = mysql_fetch_array($result)) {
                $id = $ligne[''id''];
                $name = $ligne[''name''];
                $address = $ligne[''address''];
                echo ''$id - $name, $address <br />'';
        }
} else {
        echo ''Erreur de requête de base de données.'';
}
```

Ici, on accède aux valeurs de la ligne par l'attribut dans le tableau associatif.

Sessions (I)

Les sessions sont un moyen de sauvegarder et de modifier des variables tout au cours de la visite d'un internaute sans qu'elles ne soient visibles dans l'URL et quelque soient leurs types (tableau, objet…).

Les informations de sessions sont conservées en local sur le serveur tandis qu'un identifiant de session est posté sous la forme d'un cookie chez le client (ou via l'URL si le client refuse les cookies).

Quelques fonctions :

session_start() : démarre une session

session_destroy() : détruit les données de session et ferme la session

session_register(''var'') : enregistre la variable **$var** dans la session en cours. Attention, ne pas mettre le signe **$** (dollars) devant le nom de variable

session_unregister(''var'') : détruit la variable **$var** de la session en cours

Sessions (II)

Une session doit obligatoirement démarrer avant l'envoi de toute information chez le client : donc pas d'affichage et pas d'envoi de header. On peut par exemple avoir une variable globale contenant le code HTML de la page et l'afficher à la fin du script.

Les variables de session sont accessibles comme des variables globales du script.

Exemple :

```
<html><body>
<?
session_start();                          // démarrage de la session
if(!isset($client_ip)) {
        $client_ip = $REMOTE_ADDR; /* $REMOTE_ADDR est une variable d'environnement qui permet de retourner l'adresse IP du client*/
        session_register(''client_ip'');  // enregistrement d'une variable
}
        /* … + code de la page */
echo $out;
?>
</body></html>
```