Programmation Côté Serveur

Cours – TP – Projet

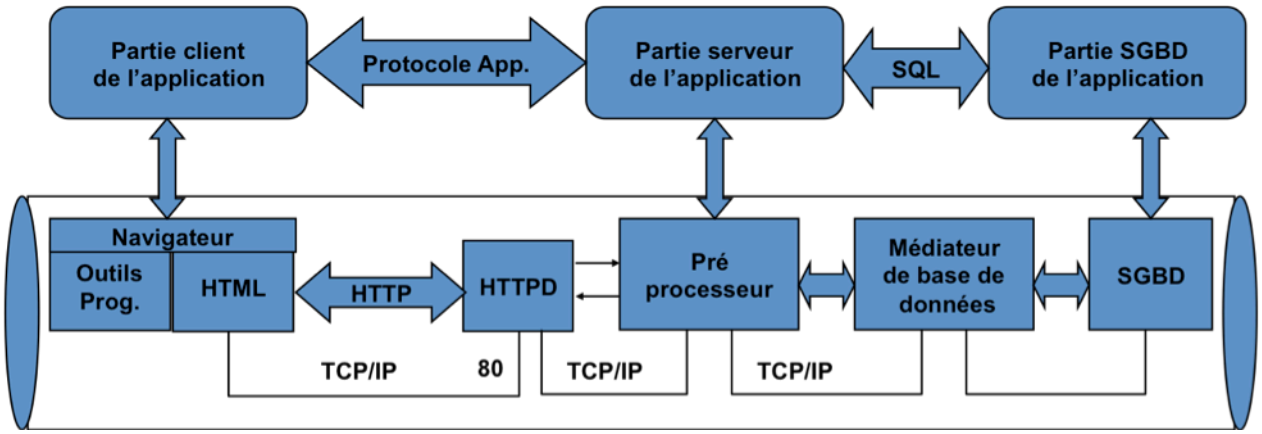
CC (0,75) + Projet (0,75) + DS (1)

Objet de ce Cours

- Développer des programmes qui s ’exécutent sur un serveur web distant à partir d’une page en html.
- Rendre une page web dynamique et traiter les données d’un formulaire html via des programmes qui s’exécutent sur un serveur distant.
- Mettre à jour des SGBD (Système de base de données) sur un serveur distant.
- Lancer des programmes destinés à effectuer des tâches de fond pour des actions répétées dans le temps.
- Utiliser les outils employés par les entreprises et apprendre à travailler dans le contexte d’un modèle MVC (Model, View, Controler)

Modèle Client/Serveur 3 Tiers

1ier Tiers | 2ième Tiers | 3ième Tiers

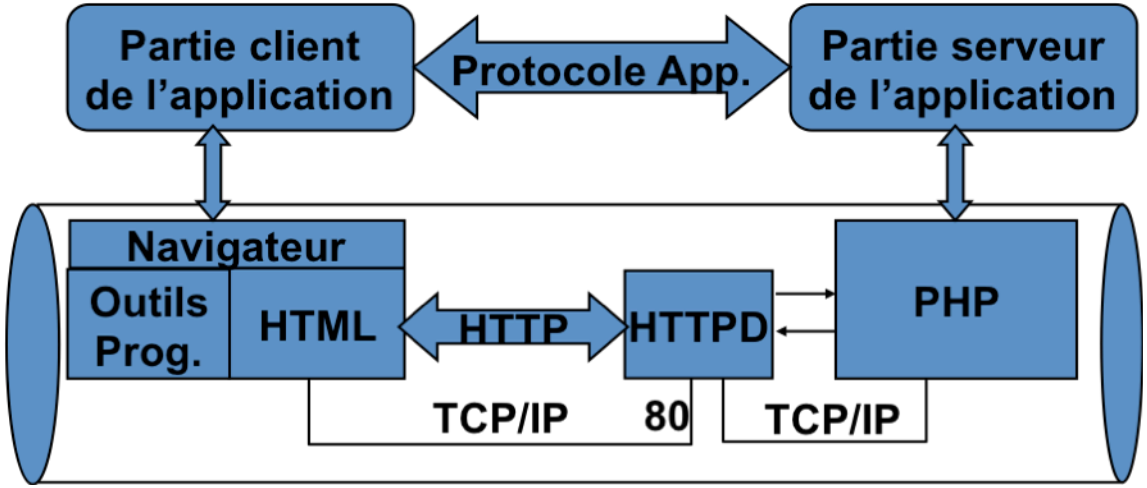


HTML, CSS
Javascript

PHP, ASP, CGI-BIN
JAVA

Oracle, Postgres
MySql, SyBase...

PHP dans un modèle C/S



Le second tiers concerne les langages opérationnels sur le serveur.

Nous allons donc nous consacrer à l'étude du langage PHP :

- Parfaite intégration avec HTML
- Permet de gérer des Bases de Données grâce à une interface SQL

Introduction au PHP

Qu'est-ce-que PHP ?

- PHP : Hypertext Preprocessor.
- Script exécuté sur le serveur comme ASP (Active Server Pages développé par Microsoft).
- PHP supporte un grand nombre de bases de données (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL…).
- PHP est un software libre d'utilisation.
- PHP peut être téléchargé et utilisé librement.
- Site officiel de php : http://www.php.net

Introduction au PHP

Qu'est-ce-qu'un fichier PHP ?

- Les fichiers PHP peuvent contenir du texte, des balises HTML et des scripts.
- Les fichiers PHP reviennent au navigateur comme des fichiers HTML.
- Les fichiers PHP sont suffixés par « .php », ou encore « .phtml ».

Introduction au PHP

Qu'est-ce MySQL ?

- MySQL est un serveur de base de données.
- MySQL convient parfaitement aux applications de petite ou de moyenne taille.
- MySQL est compatible avec les standard SQL (Structured Query Language).
- MySQL peut être compilé sur plusieurs systèmes d'exploitation (Unix, Linux, Windows, Mac OS…).
- MySQL peut être téléchargé ou utilisé librement.
- Site officiel de MySQL : http://www.mysql.com.

Introduction au PHP

PHP et MySQL

- MySQL, comme d'autres SGBD, possède une interface complète de fonctions PHP.
- PHP permet d'exécuter tout type d'opérations sur une base de données en MySQL.
- PHP s'utilise avec MySQL quelque soit le système d'exploitation, exemple :
  - Développement sous windows.
  - Exécution sur un serveur sous UNIX ou Linux.

Introduction au PHP

Pourquoi du PHP ?

- PHP est compatible avec un grand nombre de systèmes d'exploitations (Unix, Linux, Windows…).
- PHP est compatible avec la majorité des serveurs Internet utilisés de nos jours (Apache, IIS…).
- PHP est téléchargeable et utilisable librement à partir du site officiel de PHP, www.php.net qui regroupe aussi un grand nombre d'explications et d'exemples.
- PHP est développé sans cesse par une large communauté dans le monde. Dernières versions PHP 7.X
- PHP est facile à apprendre et son fonctionnement côté serveur est optimisé.

Introduction au PHP

Comment commencer à l'utiliser ?

- Pour simuler une utilisation Client/Serveur il est possible d'installer sur votre PC le logiciel « WAMP » serveur, accessible librement sous www.easyphp.org.
- Sur le site de l'IUT une application Client/Serveur est déjà installée.

Les Premiers Pas en PHP

Syntaxe de base

- Un fichier écrit en PHP est suffixé par **.php**; par exemple index.php est le fichier PHP principal du répertoire qui le contient comme index.html pour du HTML.
- Un fichier écrit en PHP est constitué par du code HTML où peut être inséré des blocs de code PHP.
- Un bloc de code en PHP commence par la balise
  - **<?php** ou parfois par <?
  - et se termine par **?>**.
- Le code PHP exécuté sur un serveur est retourné au navigateur du client sous forme de code HTML.

Les Premiers Pas en PHP

Un exemple pour commencer

On écrit le fichier monpremierfichier.php :

```
<html>
	<body>
		<center><?php	echo "Hello World";	?> </center>
	</body>
</html>
```

L'exécution de l'URL dans mon navigateur :

http://www.monsite.fr/essais/monpremierfichier.php

fait apparaître dans mon navigateur :

Hello World

NB : si mon fichier se terminait par .html (soit monpremierfichier.html), le code php n'aurait pas été exécuté.

Instructions et Commentaires

- Les instructions en PHP se terminent toujours par « ; »
- Les commentaires en PHP sont de deux formes :
  - Soit par // pour commenter une ligne
  - Soit /* et */ pour commenter un bloc de code

Les instructions pour afficher

Deux instructions permettent d'afficher des chaînes de caractères :

- echo "Hello world! ";
- print ("Hello world!");

Elles permettent toutes les deux d'interpréter directement du code html :

echo " Hello world! »;

⇒ l'affichage dans votre navigateur de :

Hello world!

Les Variables

- Déclaration d'une variable : « $ » + nom

```
$txt = "Hello World!";
$x = 16;
```

- PHP est un langage peu typé, il n'est pas nécessaire d'associer un type aux variables.
- Le nom d'une variable se compose des caractères suivants :
  - a → z,
  - A → Z,
  - 0 → 9,
  - et _
- PHP est sensible à la casse : il distingue les majuscules et minuscules dans les noms des variables.

Les Chaînes de Caractères

- Les chaînes de caractères se déclarent comme une variable :

```
<?php        $txt1 ="Bonjour tout le monde";
             $txt2= "il est dix heure"; ?>
```

- L'opérateur « . » permet la concaténation de chaînes de caractères :

```
<?php        $txt1="Bonjour tout le monde";
             $txt2="qu'allons nous faire aujourd'hui?";
             $txt3=$txt1." ".$txt2;
             echo $txt3; ?>
```

Affiche la chaîne de caractères :

Bonjour tout le monde qu'allons nous faire aujourd'hui?

- L'opérateur « .= » permet aussi la concaténation de chaînes de caractères :

```
<?php        $txt1="Bonjour tout le monde";
             $txt1.= " qu'allons nous faire aujourd'hui?";

             echo $txt1; ?>
```

Aboutit au même affichage que dans l'exemple précédent

Les Chaînes de Caractères : Fonctions

Php renferme une bibliothèque de fonctions très complètes pour traiter les chaînes de caractères; ci-dessus quelques unes :

- strlen("Hello world") renvoie comme résultat 11,
- str_word_count("Hello world") renvoie 2,
- strrev("Hello world!") renvoie !dlrow olleH,
- strpos("Hello world" , "world") renvoie 6,
- strcmp($ch1, $ch2) compare les deux chaînes,
- str_replace("world", "Dolly", "Hello world!") renvoie Hello Dolly!

Nous pratiquerons en TP les fonctions principales de traitement de chaînes de caractères.

Les Chaînes de Caractères : Caractères Spéciaux

Séquence	Signification
\n	Fin de ligne (LF ou 0x0A (10) en ASCII)
\r	Retour à la ligne (CR ou 0x0D (13) en ASCII)
\t	Tabulation horizontale (HT or 0x09 (9) en ASCII)
\v	Tabulation verticale (VT ou 0x0B (11) en ASCII) (depuis PHP 5.2.5)
\e	échappement (ESC or 0x1B (27) en ASCII) (depuis PHP 5.4.4)
\f	Saut de page (FF ou 0x0C (12) en ASCII) (depuis PHP 5.2.5)
\\	Antislash
\$	Signe dollar
\"	Guillemet double
\[0-7]{1,3}	La séquence de caractères correspondant à cette expression rationnelle est un caractère en notation octale
\x[0-9A-Fa-f]{1,2}	La séquence de caractères correspondant à cette expression rationnelle est un caractère en notation hexadécimale

Les Chaînes de Caractères : Guillemets Simples et Doubles

- Lors de l'utilisation de guillemets simples, aucun caractère spécial n'est interprété dans la chaine à part le guillemet simple :
  - echo 'Ceci n\'affichera pas \n de nouvelle ligne';
    Affiche : Ceci n'affichera pas \n de nouvelle ligne
  - echo 'Les variables ne seront pas $traitees $ici';
    Affiche : Les variables ne seront pas $traitees $ici
- Lors de l'utilisation de guillemets double, les caractères spéciaux seront interprétés dans le chaine :
  - Les variables comme $x seront remplacées par leur valeur
  - Il faudra alors veillez à précéder d'un \ les caractères spéciaux. Dans l'exemple précédent le \n entrainera un saut de ligne.

Les Chaînes de Caractères : Heredoc et Nowdoc

- La syntaxe de Heredoc est :
  $str = <<<EOD
  Exemple de chaîne
  sur plusieurs lignes
  en utilisant la syntaxe Heredoc.
  EOD;
- **Heredoc** permet donc de créer une chaîne s'étendant sur un bloc de ligne avec l'utilisation de l'opérateur <<<, d'un label (ici EOD) et du ;. De plus, PHP7 interprète la chaîne comme si elle était encadrée par des doubles guillemets.
- **Nowdoc** est aux chaînes guillemets simples ce qu'Heredoc est aux chaînes guillemets doubles, il a été mise en place à partir de PHP 5.3.4. La syntaxe est :
  echo = <<<'EOS'
  Exemple de chaîne sur plusieurs lignes en utilisant la syntaxe Nowdoc.
  Les barre oblique inversée sont toujours traité de façon litérale,
  par exemple \\ and \'.
  EOS;

Les autres types de données

- Les entiers : Integer
- Les flottants : Float ou Double
- Les Booléens : Boolean
- Les tables numériques ou associatives : Array
- Les objets : étudiés plus tard, mais nous en verrons un exemple.
- La valeur « NULL »

Les autres types de données : integer

- Varient entre :
  -2 147 483 648 et +2 147 483 647
- Ne contiennent pas de virgules
- Peuvent représenter des nombres en base 16 ou 8, ils sont alors préfixés par **0x** ou **0** :
  - 0xEF est un entier en base 16
  - 075 est un entier en base 8

Les autres types de données : float

- Ce sont les nombres équivalent aux nombres de type « float » ou « double » en C.
- Ils contiennent une virgule.
- Ils peuvent s'écrire avec un exposant soit 1,01E9 pour 1,01 milliard ou 1,01E-9 pour 1,01/1milliard.

Les autres types de données : Boolean, NULL

- Les Booléens peuvent être représentés par :
  - ➢ true ou TRUE pour vrai,
  - ➢ false ou FALSE pour faux
- Une variable a la valeur « NULL » si :
  - ➢ elle est vide, c'est à dire si elle n'a pas été affectée,
  - ➢ si elle a été affectée à NULL,
  - ➢ si elle a été effacée avec la fonction unset().

Les Constantes

- Une constante est définie par l'instruction :
  - define("UNICE", "www.unice.fr"); elle est dans ce cas sensible à la casse
  - define("UNICE", "www.unice.fr", true); elle est dans ce cas insensible à la casse, c'est à dire UNICE ou unice renverront la même valeur.
- Une constante a une définition globale : elle est vue aussi bien dans tout le fichier PHP qu'à l'intérieur des fonctions PHP que nous verrons plus tard.

Les autres types de données : Les Tables

- Deux types de tables existent en PHP :
  - Les tables avec des indices numériques.
  - Les tables associatives dont les indices sont des chaînes de caractères.
- Exemples, les instructions :
  - $cars = array("VW","BMW", "Fiat");
    et
    $cars[0] = "VW"; $cars[1]="BMW";$cars[2]="Fiat";
    sont équivalentes.
  - $ages = array("Peter"=>32, "John"=>30, "Paul"=>34);
    et
    $ages["Peter"]=32; $ages["John"]=30; $ages["Paul"]=34;
    sont équivalentes.

Les autres types de données : Les Tables

- En PHP les tables sont multidimensionnelles.
- Dans une table multidimensionnelle chaque élément de la table principale peut être une table à son tour.
- Exemple :

```
$familles = array("Audibert" =>
                        array("Pierre »,"Louis",   "Mégane"),
                  "Azoulay" =>
                        array( "Paul", "Sophie"),
                  "Bruneau" =>
                        array("Hélène", "Chloé","Audrey"));
 => $famille["Audibert"][0] vaut Pierre
 => $famille["Bruneau"][2] vaut Audrey
```

Les autres types de données : Les Tables

- L'affichage de la table précédente par l'instruction : print_r($famille) donne :

```
Array    (    [Audibert] => Array(
                        [0] => Pierre
                        [1] => Louis
                        [2] => Mégane)
              [Azoulay] => Array(
                        [0] => Paul
                        [1] => Sophie)
              [Bruneau] => Array(
                        [0] => Hélène
                        [1] => Chloé
                        [2] => Audrey))
```

Les autres types de données : Les Tables

Quelques fonctions utiles pour les tables

- explode() : Transpose les éléments d'une chaîne en ceux d'une table,
- implode() : Retourne une chaîne à partir des éléments d'une table,
- count() : Renvoie la dimension d'une table,
- str_split() : Transpose une chaîne ou les sous chaînes d'une chaîne dans une table,
- array_pop() ou array_push() : ajoute ou retranche le dernier élément d'une table,
- sort() ou asort() : ordonne une table de manière ascendante,
- rsort() ou ksort() : ordonne une table associative de manière ascendante suivant les valeurs (préfixe r) ou les clefs (préfixe k),
- arsort() ou krsort() : ordonne une table associative de manière descendante suivant les valeurs (préfixe ar) ou les clefs (préfixe kr).

Nous pratiquerons en TP les fonctions pour les tables.

Les autres types de données : Les Objets

- Un objet possède des propriétés (variables) et des méthodes (fonctions).
- Un objet est constitué par un squelette : sa classe
- Exemple de déclaration d'un objet :

```
<?php
class Car {
    function Car() {
        $this->model = "VW";
    }
}
// création d'un objet
$my_car = new Car();
// visualisation de la propriété de l'objet
echo $my_car->model;
?>
```

Les autres types de données : Variables et pseudo-types

- mixed : mixed indique qu'un paramètre peut accepter plusieurs (mais pas nécessairement tous) types.
- number : number indique qu'un paramètre peut être soit un entier, soit un nombre décimal.
- array ou object : array|object indique qu'un paramètre peut être un tableau ou un objet.
- callable : indique qu'un paramètre est de type callable.
- void : void comme type retourné signifie que la valeur retournée est inutile.

Les autres types de données : Les itérables

- Un iterable est un pseudo-type introduit en PHP 7.1. Il accepte n'importe quel tableau ou objet implémentant l'interface Traversable. Ces deux types d'itérables peuvent utiliser foreach et peuvent être appelés avec **yield from** depuis un générateur.
- On appelle une Interface Traversable, une in terface permettant de détecter si une classe peut être parcourue en utilisant foreach.
- Une fonction générateur ressemble à une fonction normale, sauf qu'au lieu de retourner une valeur, un générateur yield retourne autant de valeurs que nécessaire. Toutes fonctions contenant yield est une fonction générateur.

Les autres types de données : Les itérables

- Exemple de fonction générateur :

```
function gen_one_to_three() {
        for ($i = 1; $i <= 3; $i++) {
        // Notez que $i est préservé entre chaque production de valeur.
        yield $i;
        }
}
$generator = gen_one_to_three();
    foreach ($generator as $value) {
        echo "$value\n";
}
```

- L'exemple ci-dessus va afficher :

```
1
2
3
```

Les autres types de données : Les ressources

- Une ressource est une variable spéciale, contenant une référence vers une ressource externe. Les ressources sont créées et utilisées par des fonctions spéciales.
- Une ressource externe peut être par exemple :
  - dba_open() : lien vers une base de donnée
  - ftp_connect() : lien vers un serveur ftp

Les autres types de données : Les fonction de rappel (type callable)

- Les fonctions de rappel peuvent être identifiées via le type callable à partir de PHP 5.4.
- Une fonction PHP est passée par son nom, sous forme de chaine de caractère.
- Exemple :

```
function my_callback_function() {
    echo 'hello world!';
}
call_user_func('my_callback_function');
```

Les opérateurs en PHP

Les opérateurs arithmétique

Opérateur	Description	Exemple	Résultat
+	Addition	x=2; x+2;	4
-	Soustraction	x=2 ; 5-x;	3
*	Multiplication	x=4; x*5;	20
/	Division	15/5; 5/2;	3 2.5
%	Modulo (reste de la division)	5%2; 10%8; 10%2;	1 2 0
++	Incrément	x=5; x++;	x=6
--	Décrément	x=5; x--;	x=4

Les opérateurs en PHP

Les opérateurs d'affectation

Opérateur	Exemple	Revient à
=	x=y;	x=y;
+=	x+=y;	x=x+y;
-=	x-=y;	x=x-y;
*=	x*=y;	x=x*y;
/=	x/=y;	x=x/y;
.=	x.=y;	x=x.y;
%=	x%=y;	x=x%y;

Les opérateurs en PHP

Les opérateurs de comparaison

Opérateur	Description	Exemple
==	Est égal à	(5==8) retourne false
===	Est identique à	En plus de l'égalité précédente il faut que les éléments comparés aient le même type
!=	Est différent de (général)	(5!=8) retourne true
!==	N'est pas identique à	En plus de la différence précédente, il faut que les éléments comparés aient le même type
<>	Est différent de (chiffre)	(5<>8) retourne true
>	Est plus grand que	(5>8) retourne false
<	Est plus petit que	(5<8) retourne true
>=	Est plus grand ou égal que	(5>=8) retourne false
<=	Est plus petit ou égal que	(5<=8) retourne true

Les opérateurs en PHP

Les opérateurs logique

Opérateur	Description	Exemple
&& ou and	Et	x=6 ; y=3; (x < 10 && y > 1) retourne true
xor	Ou exclusif	x=6 ; y=3; (x < 10 xor y > 1) retourne false (x < 10 xor y < 1) retourne true
|| ou or	Ou	x=6 ; y=3; (x==5 || y==5) retourne false
!	Non	x=6 ; y=3 ; !(x==y) retourne true

Les instructions en PHP

Les expressions conditionnelles

- **if** est utilisée pour exécuter un bloc de code si certaines conditions sont réalisées.
- **if … else** est utilisée pour exécuter un bloc de code si certaines conditions sont réalisées et sinon pour exécuter un autre bloc de code.
- **if … elseif … else** est utilisée pour sélectionner l'exécution d'un bloc de code parmi plusieurs blocs de code.
- **switch** est utilisée pour choisir l'exécution d'un bloc de code parmi un grand nombre blocs de code.

Les instructions en PHP : if

On utilise cette instruction pour exécuter un bloc de code si certaines conditions sont réalisées.

La syntaxe de l'instruction if est la suivante :

```
if (conditions) {
        bloc de code;
}
```

Exemple :

```
<?php
        $d=date("D");
        if ($d=="Fri") {
                echo "Bon week-end!";      }
?>
```

Les instructions en PHP : if ... else

L'instruction if … else permet d'exécuter un bloc de code si une condition est vraie et un autre bloc de code si elle est fausse.

La syntaxe de l'instruction if … else est la suivante :

```
if (conditions) {
        premier bloc de code;
} else {
        second bloc de code;
}
```

Exemple :

```
<?php
        $d=date("D");
        if ($d=="Fri") {
                echo "Bon week-end!";    }
        else {
                echo "Bonne semaine!";
        }
?>
```

Les instructions en PHP : if … elseif … else

L'instruction if … elseif … else est utilisée pour sélectionner l'exécution d'un bloc de code parmi plusieurs blocs de code.

La syntaxe de l'instruction if … elseif … else est la suivante :

```
if (condition) {
        premier bloc de code; }
elseif {
        second bloc de code;  }
elseif {      troisième bloc de code;   }
elseif {      … … …   }

else {        dernier bloc de code ;     }
```

Les instructions en PHP : if ... elseif ... else

Exemple :

```
<?php
    $d=date("D");
    if ($d=="Fri")
            echo "Bon week-end!";
    elseif ($d=="Sun")
            echo "Bon Dimanche!";
    else
            echo "Bonne journée!";
?>
```

Les instructions en PHP : switch

L'instruction switch est utilisée pour sélectionner l'exécution d'un bloc de code parmi un grand nombre de blocs de code.

La syntaxe de l'instruction Switch est la suivante :

```
switch (n) {
    case label1:
        bloc de code à exécuter si n=label1;
        break;
    case label2:
        bloc de code à exécuter si n=label2;
        break;
    ...
    case labelk:
        bloc de code à exécuter si n=labelk;
        break;
    default:
        code à exécuter si n!=label1 et n!=label2 ... et n!=labelk;
}
```

Les instructions en PHP : switch

Exemple :

```
<?php
        switch ($x) {
            case 1:
              echo "Le chiffre 1";
            break;
            case 2:
              echo " Le chiffre 2";
            break;
            case 3:
              echo " Le chiffre 3";
            break;
            default:
              echo "Le chiffre n'est pas entre 1 et 3";
            break;
        }
?>
```

Les instructions de boucle en PHP : while

- Répète l'exécution d'un bloc d'instructions tant qu'une condition spécifique est vraie.
- Sa syntaxe est la suivante :

```
while (conditions){
        code à exécuter;
}
```

- Exemple :

```
<?php
        $i=1;
        while($i<=5) {
                echo "Le nombre est " . $i . "<br />";
                $i++;
        }
 ?>
```

Les instructions de boucle en PHP : do ... while

- Exécute toujours une première fois un bloc d'instructions.
- Vérifie la condition et répète le bloc d'instructions tant que la condition est vraie.
- Sa syntaxe est la suivante :

```
do {
        code à exécuter;
 } while (conditions);
```

- Exemple :

```
<?php
        $i=1;
        do {
                $i++;
                echo "Le nombre est " . $i . "<br />";
        } while ($i<=5);
 ?>
```

Les instructions de boucle en PHP : La boucle for

- Répète un bloc d'instructions un certain nombre de fois.
- Sa syntaxe est la suivante :

```
for (init; condition; incrément) {
    code à exécuter;
}
```

- **init** : permet d'initialiser un compteur, mais init peut être aussi du code qui sera exécuté qu'une seule fois en début de boucle.
- **condition** : évaluée à chaque itération, si l'évaluation renvoie TRUE, la boucle est réitérée, sinon elle est stoppée.
- **incrément** : incrémente le compteur de boucle; il peut aussi être une instruction qui sera exécutée en fin de boucle.

Les instructions de boucle en PHP : La boucle for

- Exemples :

```
<?php
        for ($i=1; $i<=5; $i++) {
                echo "Le nombre est " . $i . "<br /> »;
        }
?>
ou
  <?php
        for ($i=1; $i<=5; $i+=2) {
                echo "Le nombre est " . $i . "<br /> »;
        }
  ?>
  ou
  <?php
        $k = 100;
        for ($i=1; $i<=$k; $i+=2) {
                echo "Le nombre est " . $i . "<br /> »;
                if ( $i%25 == 0 ) $k/=2;
        }
  ?>
```

Les instructions de boucle en PHP : La boucle foreach

- Permet de parcourir des tables
- Sa syntaxe est la suivante :

```
foreach ($table as $valeur) {
    code à exécuter;
}
```

ou

```
foreach ($table as $clef => $valeur ) {
    code à exécuter;
}
```

Les instructions de boucle en PHP : La boucle foreach

- Exemples :

```
<?php
        $x=array("un","deux","trois");
        foreach ($x as $valeur) {
                echo $valeur . " ";
        }
?>
=> affiche :  un deux trois
<?php
        $x=array(("Peter"=>32, "John"=>30, "Paul"=>34);
        foreach ($x as $clef => $valeur) {
                echo "x[".$clef. "] = ". $valeur .  "<br/> ";
        }
?>
=> affiche :
        x[Peter] = 32
        x[John] = 30
        x[Paul] = 34
```

L’instruction break

- L'instruction *break* permet de sortir d'une structure *for*, *foreach*, *while*, *do-while* ou *switch*.
- *break* accepte un argument numérique optionnel qui vous indiquera combien de structures emboîtées doivent être interrompues.
- Exemple, break par défaut :

```
$arr = array('un', 'deux', 'trois', 'quatre', 'stop', 'cinq');
      while (list($key, $val) = each($arr)) {
       if ($val == 'stop') {
            break;    /* Vous pourriez aussi utiliser 'break 1;' ici. */
       }
       echo "$val<br />\n";
}
```

L'instruction break

- Exemple, break avec son argument optionnel :

```
$i = 0;
while (++$i) {
  switch ($i) {
   case 5:
      echo "At 5<br />\n";
      break 1;  /* Termine uniquement le switch. */
   case 10:
      echo "At 10; quitting<br />\n";
      break 2;  /* Termine le switch et la boucle while. */
   default:
       break;
   }
}
```

L'instruction continue

- Permet d'interrompre l'itération en cours dans une boucle et sauter à la prochaine itération.
- *continue* accepte un argument numérique optionnel qui vous indiquera combien de structures emboîtées doivent être sautées.
- Exemple :

```
while (list($key, $value) = each($arr)) {
   if (!($key % 2)) { // évite les membres impairs
      continue;
    }
    traite_valeurs_paires($value);
}
```

L'instruction continue

- Exemple 2, continue avec son argument optionnel :

```
$i = 0;
while ($i++ < 5) {
  echo "Dehors<br />\n";
  while (1) {
    echo "Milieu<br />\n";
     while (1) {
       echo "Intérieur<br />\n";
       continue 3;    // sort du 3e while parent soit du while principal
     }
     echo "Ceci n'est jamais atteint.<br />\n";
   }
  echo "Ceci non plus.<br />\n";
}
```

L’instruction declare

- *declare* sert à ajouter des directives d'exécutions dans un bloc de code. La syntaxe de *declare* est similaire à la syntaxe des autres fonctions de contrôle :

  declare (*directive*)
  commandes

- L'expression *directive* permet de contrôler l'intervention du bloc *declare*. Actuellement, seulement deux directives sont reconnues : la directive *ticks* et la directive d'encodage *encoding*

L’instruction declare et ticks

- Un tick est un événement qui intervient toutes les *N* commandes bas niveau tickables, exécutées par l'analyseur dans le bloc de *declare*. La valeur de *N* est spécifiée par la syntaxe ticks=*N* dans le bloc de directive *declare* :

```
declare(ticks=1);
$cpt = 0;
// Une fonction appelée à chaque événement “ticks”
function tick_handler() {
   global $cpt;
   $cpt++;
   echo "tick_handler() called : ".$cpt;
}
register_tick_function('tick_handler');
$a = 1;
if ($a > 0) {
   $a += 2;
   print($a);
}
=>
tick_handler() called 1  tick_handler() called 2  tick_handler() called 3
3
tick_handler() called 4  tick_handler() called 5
```

L’instruction declare et encode

- L'encodage d'un script peut être spécifié par script en utilisant la directive *encoding*.

- Exemple : Déclaration d'un encodage pour un script.

```
<?php
    declare(encoding='ISO-8859-1');
    // le code …
?>
```

L'instruction declare et la représentation faiblement typée

- En PHP 7 il est possible de passer à une représentation « faiblement typée » des variables.
- Ceci revient à associer comme en C un type aux variables, notamment lors des passage de variables dans les fonctions que l'on va voir ultérieurement.
- L'instruction à exécuter est pour cela :

```
<?php
        declare(strict_types=1);
        // le code …
 ?>
```

Dès que cette instruction est exécutée il faudra alors spécifier un type aux variables comme :

- int $x,
- float $v,
- string $txt,
- array $tab,
- object $car

Les instructions include et require

- L'instruction de langage *include* inclut et exécute le fichier spécifié en argument.
- La syntaxe d'appel est la suivante : include($path); ou $path est une chaîne de caractère constituée par le chemin d'accès et le nom du fichier à inclure.
- Le chemin d'accès au fichier peut être relatif ou absolu :
  include("../../myPHPlib/file1.php");
  include("/myPHPlib/file1.php");
  include("myInputFile.php");
- Si le fichier à inclure n'est pas trouvé un warning est envoyé.

Les instructions include et require

- Attention : Soyez très vigilant si vous chargez un script PHP sur un serveur distant son contenu sera exécuté sur le votre.
- L'inclusion d'un fichier PHP implique que ses fonctions et variables globales deviendront aussi celles de votre environnement de travail, c'est à dire de votre fichier PHP en cours d'exécution.
- L'instruction *require* est identique à include, sauf qu'une erreur d'exécution entraine une erreur fatale et stoppe ainsi le programme en cours.

Les instructions include_once et require_once

- Elles sont identiques à include et require.

- Sauf que si les fichiers sont déjà inclus, ils en le seront pas une seconde fois.

L’instruction return

- *return* retourne le contrôle du programme au module appelant. L'exécution reprend alors à l'endroit de l'invocation du module.
- Dans une fonction *return* termine immédiatement la fonction, et retourne l'argument qui lui est passé.
- *return* interrompt aussi l'exécution de la commande eval() ou de scripts.
- Si les parenthèses sont omises return retourne NULL

L'instruction return

- Exemples :

  Le fichier a.php contient le code :

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

  Le fichier b.php contient le code :

```
<?php
        echo "b";
        return;
?>
```

```
include("a.php");
include("b.php");
```

  ⇒ Affiche : ba

  Soit la définition de la fonction :

```
function somme(int $a, int $b) : int {
        return($a+$b);
}
somme(1,2);  retourne 3
```

L’instruction exit

- Termine le script courant.
- Les fonctions d'extinction et les destructeurs d'objets seront toujours exécutés même si *exit* est appelé.
- Syntaxe d’appel :
  void **exit** ([string $status]), affiche la chaîne $status juste avant de sortir.

  ou
  void **exit** (int $status) avec $status € [0, 255[

  NB : $status nul signifie que le code s’est terminé normalement.

L’instruction goto

- L'opérateur *goto* peut être utilisé pour continuer l'exécution du script à un autre point du programme.
- La cible est spécifiée par un label, suivi de deux-points.
- L'étiquette cible doit être dans le même contexte et fichier.
- Exemple :

```
<?php
        for($i=0,$j=50; $i<100; $i++) {
                while($j--) {
                        if($j==17) goto end;
                }
        }
        echo "i = $i";
        end:
        echo 'j hit 17';
?>
```

affiche : j hit 17

Les fonctions en PHP

- Véritable puissance de PHP, plus de 1000 fonctions sont déjà implantées.
- Nous en avons déjà découvert un grand nombre sur le site www.php.net
- Une fonction peut être appelée de n'importe quel endroit d'une page dans une séquence PHP.

Les fonctions en PHP
Créer sa propre fonction

- Syntaxe d'une fonction :

```
function nomFonction() {
        code à exécuter;
}
```

- Il est conseillé de donner à la fonction un nom qui reflète la tâche qu'elle accomplit.
- Le nom d'une fonction peut commencer par une lettre ou un « _ » mais pas par un chiffre.
- Un premier exemple :

```
function ecrireNom() {
            echo "Harry Potter";
            return();
}
echo "Mon nom est ";
ecrireNom();
```

Les fonctions en PHP : Les variables d'entrées

- Il est possible de passer des variables d'entrées par valeur ou par adresse.
- L'exemple suivant évoque le passage par valeur :

```
function add($x, $y) {
    $total=$x+$y;
    return ($total);
}
echo "1 + 16 = " . add(1,16);
=> affiche : 1 + 16 = 17
```

Les fonctions en PHP : Les variables d'entrées

- Il est aussi possible de passer les variables par adresse si l'on veut que leur valeur soit modifiée dans la fonction.
- L'exemple suivant évoque le passage par adresse :

```
function add($x,$y, &$total) {
    $total=$x+$y;
    return();
}
$total = 0;
add(1,16, &$total);
echo "1 + 16 = " . $total;

=> affiche : 1 + 16 = 17
```

Les fonctions en PHP : Les variables d'entrées

- Il est possible de fixer une valeur par défaut à une variable d'entrée.
- L'exemple suivant permet de fixer une valeur par défaut.

```
function setHeight($minheight = 50) {
    echo "La hauteur est : $minheight <br>";
    return();
}

setHeight(350);
setHeight();        // utilise dans ce cas la valeur par défaut
setHeight(135);
setHeight(80);
```

Les fonctions en PHP : Variable retournée

- L'instruction return permet de terminer une fonction sans rien retourner si elle est appelée sans variable.
- L'instruction return permet de retourner une valeur (nombre, table, chaîne de caractères) qui lui est passée en argument.
- Exemple :

```
function pv($x, $y) {
        foreach($x as $k => $v)
                $r[$k] = $v + $y[$k];
        return($r);
}
$v = array(1, 2, 3); $y = array(3, 4, 5);
$res = pv( $v, $y);
print_r($res);

=> donne le résultat :
res[0] = 4
res[1] = 6
res[2] = 8
```

Les fonctions en PHP 7 : Appel typé : variables d'entrée

- L'appel des fonctions en php7 peut être typé, dans ce cas il faut utiliser l'instruction « declare » et comme en langage C, indiquer le type des variables d'entrée et celui de la variable renvoyée.

- Exemple de type : int, float, string, boolean.

```
// instruction à exécuter pour un appel tystrict
declare(strict_types=1);
function addNumbers(int $a, int $b) : int {
    return ($a + $b);
}
echo addNumbers(5, "5 days");
// retourne une erreur car "5 days" n'est pas entier
```

Les fonctions en PHP 7 : Appel typé : variables d'entrée

- Cas de des tables :

```
function  multmatrice (array $m, array $n) : array {
        …
        return($mXn);
}
```

- Reprenons l'exemple précédent, la nouvelle formulation devient :

```
function pv(array $x,array  $y) {
      foreach($x as $k => $v)
            $r[$k] = $v + $y[$k];
      return($r);
}
```

Les fonctions en PHP 7 : Appel typé : variable renvoyée

- Dans le cas d'une fonction ne retournant rien :

```
function setVar ( … ) : void {
    … return();
}
```

- Dans le cas d'une fonction retournant une variable :

```
function somme ($int $x, int $y) : int {
        return($x + $y);
}
```

- Dans le cas d'une fonction retournant une table :

```
function pv(array $x,array  $y) : array {
    foreach($x as $k => $v)
        $r[$k] = $v + $y[$k];
    return($r);
}
```

Les fonctions « callback »

- Les fonctions en « callback » sont celles passées en argument d'une autre fonction.
- L'exemple ci dessus illustre leurs utilisations :

```
<?php
function ajout_exclamation($str) {
  return $str . "! ";
}

function ajout_interrogation($str) {
  return $str . "? ";
}

function printFormatted($str, $format) {
  // Appel de la fonction $format en tant que fonction “callback”
  echo $format($str);
}

// Passage des fonctions ajout_exclamation et ajout_interrogation
// à  printFormatted()
printFormatted("Hello world", "ajout_exclamation");
printFormatted("Hello world", "ajout_interrogation");
```

Portée des variables

- Toute variable dans une fonction qui n'est pas passée en tant que variable d'entrée est considérée comme une variable interne à la fonction.
- Les variables globales et les constantes font exception à cette règle.
- Par conséquent si une variable nommée $a est déclarée à l'extérieur d'une fonction et une autre variable nommée aussi $a est utilisée dans cette fonction, cette dernière sera différente de celle qui est à l'extérieur.
- Pour qu'une variable externe soit reconnue dans une fonction il faut la « re-déclarer » dans la fonction en la précédent de l'instruction *global*.

Portée des variables

- Exemple :

```
function testPorteeVariable () {
    global $b;
    echo "Dans la fonction la valeur de \$a est: ".$a."<br/>";
    echo "Dans la fonction la valeur de \$b est: ".$b."<br/>";
    $a = 10;
}
$a = 4; $b = 6;
testPorteeVariable ();
echo "Hors de la fonction la valeur de \$a est: ".$a;
```

=> donne comme résultat :
Dans la fonction la valeur de $a est :
Dans la fonction la valeur de $b est : 6
Hors de la fonction la valeur de $a est : 4

Portée des variables

Une variable est globale si elle est déclarée à l'aide de la table **GLOBAL** :

```
$GLOBAL["a"] = 10;
```

Le contenu de la table GLOBAL est aussi bien vu dans le corps du programme qu'à l'intérieur de la fonction qui lui fait appel.

Les fonctions « date » et « mktime »

La fonction date

- Elle retourne une chaine de caractère relative à la date et à l'heure.
- Syntaxe : date(format [, timestamp]) où :
  - format représente le format de la date,
  - timestamp, paramètre facultatif représente la date imposée; il représente le nombre de seconde entre le 01/01/1970 et la date que vous souhaitez.
- La fonction mktime() permet de calculer un timestamp, sa syntaxe d'appel est la suivante :

  mktime(heure, minute, seconde, mois, jour, année, is_dst)

  soit par exemple :

```
$demain = mktime(0,0,0,date("m"),date("d")+1,date("Y"));
```

Les fonctions « date » et « mktime »

Le format de la fonction date

- Le format de la fonction date peut être déterminé à l'aide de caractères clés.
- Par exemple les caractères comme « /», « .» ou « - » peuvent être insérés pour mieux formater la date.
- Quelques exemples de formatage sont donnés comme suit :
  - echo date("Y/m/d") donne 2015/09/22
  - date("Y.m.d ») donne 2015.09.22
  - echo date("Y-m-d") donne 2015-09-22

Les variables « superglobals »

- Des variables globales dites aussi « superglobals » sont prévues en PHP.
- Elles sont accessibles par l'ensemble des fichiers PHP.
- Ces variables sont des tables associatives, leur liste est donnée ci-dessous :
  - $GLOBAL
  - $_SERVER
  - $_REQUEST
  - $_POST
  - $_GET
  - $_FILES
  - $_ENV
  - $_COOKIE
  - $_SESSION
- Nous étudierons en détail $GLOBAL, $_SERVER, $_ENV. Les autres variables seront définies au fur et à mesure de l'avancement du cours.

Les variables « superglobals »

- $php_errormsg : Le dernier message d'erreur
- $HTTP_RAW_POST_DATA : Données POST brutes
- $http_response_header : En-têtes de réponse HTTP
- $argc : Le nombre d'arguments passés au script. Lorsque l'on exécute sur le serveur la commande php script.php arg1 arg2 arg3 => $argc vaut 3
- $argv : Tableau d'arguments passés au script. Soit par rapport à l'exemple précédent $argv[0] vaut arg1, $argv[1] vaut arg2 et $argv[2] vaut arg2;

Les variables « superglobals »: $GLOBAL

- Cette table associative permet de définir des variables globales communes à tous les fichiers PHP.

- La définition des variables se fait comme suit :
  $GLOBAL["IUTsiteName"] = "IUT – Unice »;
  $GLOBAL["PI"] = 3,1415;
  $GLOBAL["KPI"] = 10 * $GLOBAL["PI"] ;

Les variables « superglobals »: $_SERVER

- La table associative $_SERVER permet de définir l'ensemble des informations relatives aux :
  - Entêtes,
  - Chemins d'accès,
  - Emplacement de scripts.
- Nous pouvons retrouver l'ensemble des définitions sur la page : https://secure.php.net/manual/fr/reserved.variables.server.php
- Cependant les plus utilisées sont définies par :
  - $_SERVER['PHP_SELF'] : Le nom du fichier du script en cours d'exécution, la constante __FILE__ contient aussi cette valeur.
  - $_SERVER['GATEWAY_INTERFACE'] : Numéro de révision de l'interface CGI du serveur : i.e. '*CGI/1.1*'.
  - $_SERVER['SERVER_ADDR'] : L'adresse IP du serveur sous lequel le script courant est en train d'être exécuté
  - $_SERVER['SERVER_NAME'] : Le nom du serveur hôte qui exécute le script.
  - $_SERVER['DOCUMENT_ROOT'] : La racine sous laquelle le script courant est exécuté.
  - $_SERVER['HTTP_HOST'] : Contenu de l'en-tête *Host:* de la requête courante, si elle existe.
  - $_SERVER['HTTP_REFERER'] : L'adresse de la page (si elle existe) qui a conduit le client à la page courante.
  - $_SERVER['HTTP_USER_AGENT'] : Contenu de l'en-tête *User_Agent:* de la requête courante, si elle existe.
  - $_SERVER['SCRIPT_NAME'] : Contient le nom du script courant.
  - $_SERVER['SCRIPT_URI'] : Retourne l'URI de la page courante.

Les variables « superglobals »: $_ENV

- La table $_ENV contient les valeurs des variables d'environnement au niveau du serveur.
- Les variables d'environnement sont aussi récupérables via la fonction getenv().
- Exemple : prenons le cas d'un environnement shell du compte « manager » où un extrait des variables d'environnement est donné par :
  - LANG=fr_FR.UTF-8
  - USER=manager
  - LOGNAME=manager
  - HOME=/home/manager
  - PATH=/home/manager/sbin:/home/manager/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X11
  - MAIL=/var/mail/manager
  - SHELL=/bin/tcsh

  =>
  $_ENV['LANG'] vaut fr_FR.UTF-8
  $_ENV['USER'] vaut manager
  $_ENV['HOME'] vaut /home/manager
  $_ENV['SHELL'] vaut /bin/tcsh

Les expressions régulières : La fonction de filtrage : RegEx

- Une expression régulière est une suite de caractères qui constitue un critère de recherche pour un modèle donné.
- Syntaxe : $exp = "/Ceci est un exemple/i";
  - / est un délimiteur,
  - « i » est un critère qui indique que la recherche doit être sensible à la casse.

Le traitement des formulaires HTML

- Rappel : un formulaire HTML permet de saisir des données et de les transmettre à un serveur chargé de les traiter.
- Un exemple simple :

```
<form action="welcome.php" method="post">
        Nom: <input type="text" name="nom" />
        Age: <input type="text" name="age" />
        <input type="submit" name="Valider" />
</form>
```

- Lorsqu'un utilisateur renseigne les champs de ce formulaire et clique sur le bouton valider, les données sont envoyées au fichier PHP appelé welcome.php.
- Il est bon de rappeler que le mode de transmission des données dépend de la méthode :
  - Si la méthode est de type "post" la transmission se fait au travers du flux de données des fichiers d'entrée sortie entre le client et le serveur.
  - Si la méthode de transmission est de type "get" la transmission se fait au niveau de l'URI.
- L'exécution sur le serveur du fichier « welcome.php » traitera ces données et provoquera les actions voulues.

Le traitement des formulaires HTML

Où sont récupérées les données provenant d'un formulaire ?

- Ce sont les variables « superglobales » $_POST ou $_GET qui récupèrent les données du formulaire selon le type de méthode utilisée :
  - Si la balise « form » est configurée avec l'attribut « method » affecté à « GET », ce sera la variable $_GET qui contiendra les données.
  - Si la balise « form » est configurée avec l'attribut « method » affecté à « POST », ce sera la variable $_POST qui contiendra les données.
- Les variables $_POST et $_GET sont des tables associatives.
- Ainsi dans notre exemple précédent nous avions :

  <form action="welcome.php" **method="post"**>
  =>
  C'est la variable $_POST qui récupérera les données du formulaire sous la forme :
  - $_POST["nom"] **contient la valeur du champ "nom"**,
  - $_POST["age"] **contient la valeur du champ "age"**
- Nous remarquons par ailleurs que le contenu de l'attribut « name » de la balise « input » est la **clé de la donnée dans la table $_POST**.

Le traitement des formulaires HTML

Comment sont traitées les valeurs récupérées d'un formulaire?

- Dans l'exemple précédent le fichier « welcome.php » était chargé de traiter les valeurs.
- Le contenu du fichier welcome.php peut être par exemple :

```
<html>
    <body>
        Bonjour <?php echo $_POST["nom"]; ?>,
        avez vous <?php echo $_POST["age"]; ?> ans ?.
    </body>
</html>
```

- Nous voyons que les valeurs des données du formulaires y sont directement traitées comme des variables PHP.

Le traitement des formulaires HTML

Cas de la méthode de transmission « get ».

- Reprenons l'exemple précédent avec la méthode « get » :

```
<form action="welcome.php" method="get”>
        Nom: <input type="text" name="nom" />
        Age: <input type="text" name="age" />
        <input type="submit" name="Valider" />
</form>
```

- Lorsque le bouton « submit » est actionné, soit le formulaire soumis, nous verrons l'URI se transformer :
  - De : $http://www.monsite.com/welcome.php
  - En : $http://www.monsite.com/welcome.php**?nom=Kevin&age=19**, si les « input » nom et age ont été affectés respectivement en « Kevin » et « 19 ».
- Les données du formulaire par la méthode « get » se transmettent donc au travers de l'URI.
- Cette méthode de transmission qui semble pratique au premier abord, présente cependant deux inconvénients :
  - Le nombre de données à passer est limité.
  - Le fait que les valeurs des données pose des problèmes de sécurité dans le cas de variable critique de type mot de passe ou numéro de carte bleue... Donc à éviter impérativement.

Le traitement des formulaires HTML

- Le traitement des données avec la méthode « get » et identique à celle de la méthode « post ».
- Dès l'activation du formulaire, la table associative $_GET contiendra les variables :
  - $_GET["nom"] avec la valeur « Kevin »,
  - $_GET["age"] avec la valeur 19.
- Le traitement des variables dans le fichier « welcome.php » se fera alors de manière identique à ce que l'on a vu pour la méthode « post ».
- La table associative superglobale $_REQUEST contient à la fois les données transmises par la méthode « post » que celles transmises par la méthode « get ».

Le traitement des formulaires HTML

La forme des données transmises selon le type des balises « input », « select » et « textarea ».

- Pour la balise input chaque bouton, zone de texte, bouton radio ou case à cocher renvoie une valeur et une seule par élément créé.
- Il en est de même pour la balise textarea qui renvoie un texte.
- Le problème vient de la balise select avec une sélection multiple, dans ce cas la valeur renvoyée peut être unique ou multiple. Pour éviter les confusions la déclaration du select doit être faite comme suit :

```
<select name="cars[]" multiple>
    <option value="volvo">Volvo</option>
    <option value="vw">VW</option>
    <option value="bmw">BMW</option>
    <option value="audi">Audi</option>
</select>
```

- Dans ce cas la valeur de $_POST["cars"] ou $_GET["cars"] sera automatiquement assimilée à une table.

Le traitement des formulaires HTML

La variable PHP_SELF et l'analyse des champs des formulaires

- Le code pour des fichiers de formulaires qui s'appelle eux même est le suivant :

```
<form method="post"
    action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

- Il est conseillé de **tester systématiquement les variables** en provenance des formulaires car les « hackers » se font un plaisir d'insérer du code dans les variables pour pirater un site. La fonction à utiliser peut être la suivante :

```
function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

  - La fonction trim() supprime les espaces, saut de lignes, tabulations...)
  - La fonction stripslashes() supprime les backslashes : « \ »
  - La fonction htmlspecialchars() supprime les caractères codés en HTML comme < , > ...

- Les **champs obligatoires doivent jamais être vide**, la fonction PHP qui permet de tester si le champs d'un formulaire est bien rempli est empty() :

```
if (empty($_POST["email"])) {
    $emailErr = "Vous devez indiquer votre adresse électronique";
} else {
    $email = test_input($_POST["email"]);
}
```

Le traitement des formulaires HTML

La variable l'analyse des champs des formulaires

Pour analyser le contenu des champs d'un formulaire nous pouvons faire appel aux fonctions :

- preg_match() qui permet de ne conserver que des caractères souhaités, comme par exemple que des lettres minuscules, des « . » ou « @ ».
  Exemple pour un nom : il ne doit contenir que des minuscules, majuscules, des espaces ou des « - » :

```
if (!preg_match("/^[a-zA-Z\-]*$/",$nom)) {
    $nameErr = "Seul les lettres, espaces et tirets sont admis";
}
```

- filter_var qui est une fonction de filtrage de chaîne de caractère.
  Exemple pour un email :

```
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $emailErr = "Format de l\'email invalide";
 }
```

Formulaire : un exemple

```
Nom: <input type="text" name="name" value="<?php echo $name;?>">

E-mail: <input type="text" name="email" value="<?php echo $email;?>">

Website: <input type="text" name="website" value="<?php echo $website;?>">

Commentaire: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>

Civilité </br> :
Femme <input type="radio" name="civ"
                <?php if (isset($civ) && $civ=="femme") echo "checked";?>
                value="femme">
Homme <input type="radio" name="civ"
                <?php if (isset($civ) && $civ=="male") echo "checked";?>
                value="homme”>
Autre    <input type="radio" name="civ"
                <?php if (isset($civ) && $civ=="civ") echo "checked";?>
                value="autre”>
```

```php
<?php
// définition des valeurs
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $civ = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Un nom est requis";
  } else {
    $name = test_input($_POST["name"]);
    // vérifier si le nom contient que des lettres
    if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Un email est requis";
  } else {
    $email = test_input($_POST["email"]);
    // vérifier si l'adresse email est correctement formulée
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    //vérifier sir l'URL est valide  (l'expression régulière autorise les / dans l'URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST[”civ"])) {
    $genderErr = "La civilité est requise";
  } else {
    $civ = test_input($_POST["civ"]);
  }
}
```

Téléchargement de fichiers

Téléchargement ou « Upload » de fichier

- Les formulaires permettent le téléchargement de fichiers du client sur le serveur.
- Le téléchargement de fichier étant une opération critique par rapport à la sécurité et par rapport à la place occupée sur le serveur de nombreuses précautions doivent être prises.
- Rappel : Le téléchargement de fichier est possible via la balise « input » d'un formulaire dont l'attribut « type » est affecté à la valeur « file ».
- Remarque : Pour accepter les téléchargements de fichier sur le serveur, le fichier de configuration globale de PHP php.ini doit contenir l'instruction :

  file_uploads = On

Téléchargement de fichiers

- Exemple de téléchargement de fichier :

```
<form action="upload_file.php" method="post”
        enctype="multipart/form-data”>
    <label for="file">Nom du fichier:</label>
    <input type="file" name="file" id="file" />
    <br />
    <input type="submit" name="submit" value="Valider" />
</form>
```

- L’utilisation de l’attribut « enctype » dans la balise <form> permet de spécifier le type du contenu du fichier à télécharger.

- La valeur « multipart/form-data » permet d’indiquer que le formulaire utilise des données binaires.

Téléchargement de fichiers

La table superglobale $_FILE

- C'est la table associative superglobale $_FILES qui permet de gérer le téléchargement.

- La table associative $_FILES est décrite comme suit :
  - $_FILES["file"]["name"] indique le nom du fichier téléchargé,
  - $_FILES["file"]["type"] indique le type du fichier téléchargé,
  - $_FILES["file"]["size"] indique la taille du fichier téléchargé,
  - $_FILES["file"]["tmp_name"] indique le nom du fichier temporaire copié sur le serveur,
  - $_FILES["file"]["error"] indique le code d'erreur résultant du téléchargement, s'il y en a une.

Téléchargement de fichiers

Création d'un script de téléchargement

- Un premier script de téléchargement peut être programmé comme suit :

```
if ($_FILES["file"]["error"] > 0) {
  echo "Erreur: " . $_FILES["file"]["error"] . "<br />";
} else {
  echo "Téléchargement de : " . $_FILES["file"]["name"] . "<br />";
  echo "Type : " . $_FILES["file"]["type"] . "<br />";
  echo "Taille : " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
  echo "Stocké dans : " . $_FILES["file"]["tmp_name"];
}
```

Téléchargement de fichiers

Les restrictions sur le téléchargement

- Nous pouvons restreindre la taille et le type des fichiers téléchargés.
- L'exemple suivant illustre un script de téléchargement pour une image de type « jpeg » dont la taille doit être inférieure à 20Kb :

```
if ((($_FILES["file"]["type"] == "image/gif")
                || ($_FILES["file"]["type"] == "image/jpeg")
                || ($_FILES["file"]["type"] == "image/pjpeg"))
            && ($_FILES["file"]["size"] < 20000)) {
           if ($_FILES["file"]["error"] > 0) {
                echo "Erreur: " . $_FILES["file"]["error"] . "<br />";
           } else {
                echo "Fichier: " . $_FILES["file"]["name"] . "<br />";
                echo "Type: " . $_FILES["file"]["type"] . "<br />";
                echo "Taille: " . ($_FILES["file"]["size"] / 1024) . "Kb<br />";
                echo "Stocké dans: " . $_FILES["file"]["tmp_name"];
           }
      } else {
                echo "Fichier invalide";    }
```

- Nous remarquons que le fichier est stocké dans un répertoire temporaire . $_FILES["file"]["tmp_name"], qui en général porte le nom "tmp" sous la racine de votre serveur.

Téléchargement de fichiers

Téléchargement simultanés de plusieurs fichiers

- Considérons le script suivant :

```
<form action="file-upload.php" method="post" enctype="multipart/form-data">
    Envoyez plusieurs fichiers : <br />
    <input name="userfile[]" type="file" /><br />

    <input type="submit" value="Envoyer les fichiers" />
</form>
```

- Nous voyons que l'attribut name est affecté à une table $userfile[]; si on suppose que les fichiers *home/test/review.html* et */home/test/xwp.out* ont été téléchargés, alors :
  - *$_FILES['userfile']['name'][0]* contient *review.html* et *$_FILES['userfile']['name'][1]* contient *xwp.out*
  - *$_FILES['userfile']['size'][0]* va contenir la taille du fichier *review.html*, etc.
- **Remaque :** la variable de configuration PHP max_file_uploads fixe le nombre maximal de fichiers téléchargeables simultanément.

Téléchargement de fichiers

Sauvegarde du fichier téléchargé

- Les fichiers téléchargés sont dans un premier temps copiés dans le répertoire temporaire généralement nommé « tmp » créé pour cet usage sous la racine de votre site web.
- Dès que le téléchargement est traité le fichier temporaire doit être déplacé par un « move » dans son répertoire de destination de votre site web.
- En supposant que le fichier téléchargé a passé la validation comme dans le transparent précédent, un exemple de script peut être :

```
if (file_exists("upload/" . $_FILES["file"]["name"])) {
  echo $_FILES["file"]["name"] . " existe déjà. ";
} else {
  move_uploaded_file($_FILES["file"]["tmp_name"],
          "upload/" . $_FILES["file"]["name"]);
  echo "Stocké dans: " . "upload/" . $_FILES["file"]["name"];
}
```

- Dans l'exemple ci-dessus notre répertoire de destination se nomme "upload".

Manipulation de fichier sur serveur

Nous traitons dans cette partie du cours les fonctions d'ouverture, de fermeture, de lecture et d'écriture de fichiers se trouvant sur le serveur.

Ouverture d'un fichier

- Syntaxe :
  fopen("filepath/filename", "mode") ;
- filepath représente le chemin d'accès au fichier,
- mode représente le mode d'ouverture du fichier,
- La fonction fopen() retourne 0 dans le cas où le fichier n'existe pas.
- Exemple : Ouverture du fichier welcome.txt sous le répertoire courant en mode lecture :
  (file=fopen("welcome.txt", "r")) or exit("Impossible d'ouvrir le fichier!");

Manipulation de fichier sur serveur

La table suivante résume l'ensemble des appels à la fonction fopen() :

Modes	Description
r	Mode lecture. Commence du début du fichier.
r+	Mode lecture/ecriture. Commence du début du fichier.
w	Mode écriture. Ouvre et efface le contenu d'un fichier, dans le cas où le fichier n'existe pas, un nouveau fichier est créé.
w+	Mode lecture/ecriture. Ouvre et efface le contenu d'un fichier, dans le cas où le fichier n'existe pas, un nouveau fichier est créé.
a	Mode ajout. Ouvre un fichier et écrit à la fin du fichier, dans le cas où le fichier n'existe pas, un nouveau fichier est créé.
a+	Mode lecture et ajout. Ecrit à partir de la fin du fichier en préservant son contenu.
x	Mode création. Crée un nouveau fichier, retourne FALSE si le fichier existe déjà.
x+	Mode lecture et création. Crée un nouveau fichier, retourne FALSE si le fichier existe déjà.

Manipulation de fichier sur serveur

- La fonction fclose() est utilisée pour fermer un fichier.
- Il faut toujours refermer un fichier ouvert.
- La syntaxe de la fonction fclose() est la suivante :

```
$file = fopen("test.txt","r");
//code à executer …
fclose($file);
```

Manipulation de fichier sur serveur

Les fonctions de de lecture

- La fonction feof() permet de vérifier si le pointeur de lecture ou écriture est en fin de fichier :
  => feof($file) retourne TRUE si la fin du fichier est atteint
- La fonction fgets() permet de lire une ligne d'un fichier.
- L'appel de fgets() déplace le pointeur de lecture du fichier à la ligne suivante.
- Exemple :

```
$file = fopen("welcome.txt", "r") or exit("Impossible d'ouvrir le fichier!");
//Lecture de la ligne suivante tant que la fin n'est pas atteinte
while(!feof($file)) {
    echo fgets($file). "<br />";
}
fclose($file);
```

Manipulation de fichier sur serveur

Les fonctions de lecture

- La fonction fgetc() permet de lire un caractère dans un fichier.
- La fonction fgetc() déplace le pointeur de lecture du fichier au caractère suivant.
- L'exemple qui suit illustre l'utilisation de fgetc():

```
$file = fopen("welcome.txt", "r") or exit("Impossible d'ouvrir le fichier!");
//Lecture du caractère suivant tant que la fin n'est pas atteinte
while(!feof($file)) {
        echo fgetc($file). "<br />";
}
fclose($file);
```

Manipulation de fichier sur serveur

Les fonctions d'écriture

- La fonction fwrite() est utilisée pour écrire dans un fichier.
- Exemple :

```
$file = fopen("welcome.txt", "w") or exit("Impossible d'ouvrir le fichier!");
// Ecriture d'un texte
fwrite($file, "Bienvenue au département informatique\n");
fwrite($file,  "Nous étudions le PHP\n");
fclose($file);
```

=> le fichier welcome.txt contient :

```
Bienvenue au département informatique
Nous étudions PHP
```

- Si l'on exécute à nouveau le script précédent avec :

```
fwrite($file, "Bienvenue au département informatique\n");
fwrite($file,  "Nous étudions le PHP\n");
```

=> les phrases précédemment entrées vont être écrasées.

Manipulation de fichier sur serveur : Les groupes de fonctions en détail

Nous allons maintenant parcourir les groupes de fonctions principales pour manipuler des fichiers en dehors de celles vues précédemment.

Modification des droits des fichiers

- Les fonctions pour modifier les droits des fichiers sont inspirées des commandes shell :
  - chgrp() : change le groupe d'un fichier,
  - chmod() : modifie les propriétés (lecture, écriture, exécution...) d'un fichier,
  - chown() : modifie le propriétaire d'un fichier,
  - umask() : change les permissions d'un fichier,
  - lchgrp() : change le groupe d'un lien,
  - lchown() : change le propriétaire d'un lien.

Effacement des fichiers et manipulation du cash

- Les fonctions pour effacer un fichier et vider le cash sont les suivantes :
  - Effacement : delete(), unlink() ou unset(),
  - Vidange du cash : clearstatcache(), fflush(),
  - Information sur le cash : realpath_cache_get(), realpath_cache_size().

Manipulation de fichier sur serveur : Dates et Droits

Informations sur le fichier

- Les fonctions pour obtenir les informations relatives à la modification d'un fichier sont les suivantes :
  - fileatime() : date de dernier accès au fichier,
  - filectime() : date de dernier du changement du fichier,
  - filemtime() : date de dernière modification du contenu fichier,
  - touch() : permet de modifier la date d'un fichier.

- Les fonctions relatives au droits et propriétés du fichiers sont les suivantes :
  - filegroup() : retourne l'ID du groupe du fichier,
  - fileowner() : retourne le nom du propriétaire du fichier,
  - fileperms() : retourne le statut des permissions du fichier,
  - filesize() : retourne la taille d'un fichier,
  - filetype() : retoune le type d'un fichier => file, link, dir,
  - fstat() : retourne les informations sur un fichier ouvert,
  - ftell() : retourne la position courante d'un fichier,
  - pathinfo() : retourne les informations relatives au path d'un fichier,
  - Information sur le cash : realpath_cache_get(), realpath_cache_size().

Manipulation de fichier sur serveur : Etats

Check des états des fichiers

- is_executable() : retourne TRUE si fichier exécutable,
- is_file() : retourne TRUE si c'est un fichier,
- is_link() : retourne TRUE si c'est un lien,
- is_readable() : retourne TRUE si le fichier est lisible,
- is_writable() ou is_writeable() : retourne TRUE si le fichier peut être ouvert en mode écriture.
- is_uploaded_file() : retourne TRUE si le fichier a été téléchargé.

Manipulation de fichier sur serveur : Création, Fermeture, Cash

L'ouverture, la création ou la fermeture d'un fichier ou d'un lien sont gérés par les fonctions qui suivent :

- Création :
  - fopen() : ouverture d'un fichier,
  - copy() : copie d'un fichier,
  - tmpfile() : création d'un fichier temporaire,
  - link() : création d'un lien logique,
  - popen() : création d'un fichier à l'issue d'un « pipe ».
- Fermeture :
  - fclose() : fermeture d'un fichier,
  - pclose() : fermeture d'un « pipe ».
- Effacement :
  - delete(), unlink() ou unset() : effacement d'un fichier ou d'un lien.

PHP met en mémoire cash des données d'un fichier pour plus d'efficacité, si nécessaire, pour vider le cash on utilise :

- clearstatcache() : efface les données enregistrées en mémoire concernant les caractéristiques d'un fichier (groupe, propriétaire, taille, id, en lecture ou écriture ...),
- fflush() : efface les données du fichier stockées en cash.

Manipulation de fichier sur serveur : Lecture

Lecture dans un fichier

- fgetc() : Retourne le caractère au niveau du curseur de fichier,
- fgetcsv() : Récupère une ligne d'un fichier csv,
- fgets() : Récupère une ligne d'un fichier,
- fgetss() : Récupère une ligne d'un fichier sans les tags HTML et PHP,
- file() : Lit un fichier et le recopie dans une table,
- file_get_contents() : Copie un fichier dans une chaine,
- fpassthru() : Lit un fichier jusqu'à sa fin et le copie sur le buffer d'un fichier ouvert,
- fread() : Lecture à partir d'un fichier ouvert,
- fscanf() : Lecture à partir d'un fichier ouvert selon un format spécifié.
- readfile() : Lit un fichier, affiche son contenu et retourne le nombre d'octets (caractères) qu'il contient .

Manipulation de fichier sur serveur : Positionnement du curseur

Les fonctions pour placer ou lire le curseur de pointage dans un fichier sont :

- fseek() : va à la position indiquée par le curseur,
- ftell() : donne la place du curseur dans le fichier,
- rewind() : remet le curseur en début de fichier,
- feof() : teste si la fin du fichier est atteint.

Manipulation de fichier sur serveur : Environnement

- basename() : retourne le nom du fichier sans son path complet,
- dirname() : retourne le nom du répertoire en cours,
- disk_free_space() ou diskfreespace() : retourne la taille restante du disque,
- disk_total_space() : retourne la taille du disque,
- glob() : retourne la liste des fichiers et répertoires dans une table,
- mkdir() ou rmdir() : crée un répertoire ou efface un répertoire vide,
- pathinfo() : retourne une table contenant les informations sur le path,
- realpath() : retourne le path avec le chemin absolu,
- rename() : renomme un fichier ou un répertoire.

Manipulation de fichier sur serveur

- La liste complète de toutes les fonctions possibles pour manipuler un fichier se trouve sous :
  - http://fr.php.net/manual/fr/ref.filesystem.php
  - http://www.w3schools.com/php/php_ref_filesystem.asp

Manipulation des répertoires

PHP comporte plusieurs fonctions pour manipuler les répertoires :

- opendir(), closedir() : ouverture et fermeture du répertoire dont le path est passé en variable; opendir() retourne un pointeur sur le répertoire ouvert.
- chdir() : va au répertoire passé en path et retourne un pointeur sur ce répertoire.
- readdir() : renvoie la donnée pointée par le pointeur de répertoire et fait avancer d'un cran le pointeur de répertoire.
- scandir() : renvoie le contenu d'un répertoire passé en path sous forme d'une table.

Manipulation des répertoires

- dir() : retourne une instance de l'objet « directory ».
- chroot() : modifie le répertoire racine.
- getcwd () : retourne le répertoire courant.
- rewinndir() : repositionne le pointeur de répertoire en début du répertoire.

Manipulation des répertoires : exemples

Quelques exemples...

```
$dir = "/images/";

// Open a directory, and read its contents
if (is_dir($dir)){
  if ($dh = opendir($dir)){
    while (($file = readdir($dh)) !== false){
      echo "filename:" . $file . "<br>";
    }
    closedir($dh);
  }
}
?>
```

Donne :

filename: cat.gif
filename: dog.gif
filename: horse.gif

Manipulation des répertoires : exemples

```
$dir = "/images/”;

// Par défaut renvoie le contenu dans un ordre ascendant
$a = scandir($dir);
print_r($a);
```

Donne :

```
Array
(
[0] => .
[1] => ..
[2] => cat.gif
[3] => dog.gif
[4] => horse.gif
[5] => myimages
)
```

Outils de transfert de fichier : FTP

Pour que PHP intègre les outils de transfert FTP, il doit être compilé sur le serveur avec l'option « --enable-ftp ».

- Sous PHP 5 de nombreuses fonctions et constantes sont mises en place pour gérer le protocole FTP.
- Ces fonctions sont similaires à celles utilisées par le protocole ftp. Nous allons en étudier quelques unes. Vous pourrez cependant trouver l'ensemble de ces fonctions sous :
  - http://php.net/manual/fr/ref.ftp.php
  - http://www.w3schools.com/php/php_ref_ftp.asp

Outils de transfert de fichier : FTP

Les principales fonctions ftp sous PHP :

- ftp_connect() : Ouvre une connexion FTP.
- ftp_ssl_connect() : Ouvre une connexion FTP sécurisée avec SSL.
- ftp_login() : Identification sur un serveur FTP (serveur/ user/passwd).
- ftp_close() ou ftp_quit() : Ferme une connexion FTP.
- ftp_put() : Charge un fichier sur un serveur FTP.
- ftp_get() : Télécharge un fichier depuis un serveur FTP.

Outils de transfert de fichier : FTP

Les principales fonctions ftp sous PHP :

- ftp_rename() : Renomme un fichier sur un serveur FTP.
- ftp_delete() : Efface un fichier sur un serveur FTP.
- ftp_mkdir() / ftp_rmdir() : Créé / Efface un dossier FTP.
- ftp_chdir() : Modifie le dossier FTP courant.
- ftp_exec() : Exécute une commande sur un serveur FTP.

Outils de transfert de fichier : FTP

Les constantes pré-définies peuvent être retrouvées sous :

- http://php.net/manual/fr/ftp.constants.php
- http://www.w3schools.com/php/php_ref_ftp.asp

Les principales constantes pré-définies sont :

FTP_ASCII	**entier**
FTP_TEXT	entier
FTP_BINARY	entier
FTP_IMAGE	entier
FTP_TIMEOUT_SEC	entier

Outils de transfert de fichier : FTP

Quelques exemples ...

```
// Définition de quelques variables
$local_file = 'local.zip';
$server_file = 'server.zip';

// Mise en place d'une connexion basique
$conn_id = ftp_connect($ftp_server);

// Identification avec un nom d'utilisateur et un mot de passe
$login_result = ftp_login($conn_id, $ftp_user_name, $ftp_user_pass);

// Tentative de téléchargement du fichier $server_file et sauvegarde dans le fichier $local_file
if (ftp_get($conn_id, $local_file, $server_file, FTP_BINARY)) {
   echo "Le fichier $local_file a été écris avec succès\n";
} else {
   echo "Il y a un problème\n";
}

// Fermeture de la connexion
ftp_close($conn_id);
```

Outils de transfert de fichier : FTP

Quelques exemples ...

```
// Initialisation de la variable
$command = 'ls -al > files.txt';

// Initialisation de la connexion
$conn_id = ftp_connect($ftp_server);

// Identification
$login_result = ftp_login($conn_id, $ftp_user_name, $ftp_user_pass);

// Exécution d'une commande
if (ftp_exec($conn_id, $command)) {
   echo "$command a été exécuté avec succès\n";
} else {
   echo "Impossible d\'exécuter : $command\n";
}

// Fermeture de la connexion
ftp_close($conn_id);
```

PHP et les archives ZIP

Le serveur doit posséder les librairies nécessaires pour manipuler les archives zip. Comme précédemment nous allons voir les fonctions principales relatives à l'ouverture et la lecture d'une archive zip :

- zip_open(), zip_close() : ouvre et ferme une archive zip.
- zip_entry_open(), zip_entry_close() : ouvre et ferme un dossier d'archive.
- zip_read() : Lit la prochaine entrée dans une archive ZIP.
- zip_entry_read() : Lit le contenu d'un fichier dans un dossier.

PHP et les archives ZIP

- zip_entry_compressedsize() : Lit la taille compressée d'un dossier d'archives.
- zip_entry_compressionmethod() : Lit la méthode de compression utilisée sur un dossier d'archives (ex: rar, bz2, zip, tgz…).
- zip_entry_filesize() : Lit la taille décompressée d'un dossier d'archives.
- zip_entry_name() : Lit le nom d'un dossier d'archives.

PHP et les archives ZIP : exemples

```
$zip = zip_open("test.zip");

if ($zip) {
	while ($zip_entry = zip_read($zip)) {
		echo "Nom: " . zip_entry_name($zip_entry) . "<br />";
	}
	zip_close($zip);
}
```

Donne :

Nom: index.php
Nom: TP1
...

PHP et les archives ZIP : exemples

```
$zip = zip_open("test.zip");

if ($zip)  {
      while ($zip_entry = zip_read($zip)) {
            echo "<p>";
            echo "Nom: " . zip_entry_name($zip_entry) . "<br />";
             if (zip_entry_open($zip, $zip_entry)) {
                  echo "Contenu du fichier:<br/>";
                  $contents = zip_entry_read($zip_entry);
                  echo "$contents<br />";
                  zip_entry_close($zip_entry);
            }
            echo "</p>";
      }
      zip_close($zip);
}
```

PHP et les archives ZIP : exemples

Donne :

Nom: ziptest.txt
Contenu du fichier:
Hello World! Test pour archivage zip.

Nom: htmlziptest.html
Contenu du fichier: **Hello World!**

Test pour archivage zip en PHP.

Les Cookies PHP

Qu'est-ce qu'un « cookie » ?

- Un cookie est une variable qui est stockée dans votre navigateur et qui a une durée de vie limitée dans le temps.
- C'est donc une sorte de variable globale qui est visible par tout votre site en PHP.
- Un cookie n'est affecté que si le navigateur de l'utilisateur est configuré pour accepter les cookies.

A quoi sert un « cookie » ?

- Un cookie est utilisé pour se rappeler d'une variable lorsqu'un site est revisité régulièrement.
- Un cookie peut aussi servir à un site de stocker des données sur l'utilisateur du site sans qu'il en soit conscient (si votre navigateur accepte les cookies bien entendu).

Les Cookies PHP

Utilisation d'un cookie

- Chaque fois que l'ordinateur du client vient se connecter au serveur, les données contenues dans le cookie sont communiquées au serveur.
- La fonction sekcookie() permet de créer un cookie et de lui affecter des valeurs.
- La fonction setcookie() permet aussi de mettre à jour un cookie déjà créé.
- **Remarque :** La fonction setcookie() doit être appelée **avant** la balise <html>
- Syntaxe :

  setcookie(nom, valeur, expire, path, domaine);

Les Cookies PHP

- Exemples d'utilisation de la fonction setcookie :
  - Cas 1 : Création d'une variable « utilisateur » avec la valeur « Paul Durant » et dont la durée de vie est 1 heure : **setcookie("utilisateur", "Paul Durand", time()+3600);**
  - Cas 2 : Création d'une variable « utilisateur » avec la valeur « Paul Durant » et dont la durée de vie est 30 jours : setcookie("utilisateur", "Paul Durand", time() +**60*60*24*30**);
- Remarques :
  - La valeur d'un cookie est codée urlencode().
  - Si l'on veut éviter ce codage il faut utiliser la fonction setrawcookie()

Les Cookies PHP

Comment récupérer un cookie?

- La variable super globale $_COOKIE est utilisée pour récupérer la valeur d'un cookie.
- Exemple, récupération du cookie « utilisateur » créé précédemment :

```
if (isset($_COOKIE["utilisateur"]))
  echo "Bienvenue " .$_COOKIE["utilisateur"] . "!<br />";
else
  echo "Bienvenue <br />";
```

Les Cookies PHP

Comment supprimer un cookie?

- Pour supprimer un cookie, nous devons nous assurer que sa date d'expiration est passée.
- Exemple, suppression du cookie « utilisateur » :

  setcookie("utilisateur", "", time()-3600);

  => la suppression consiste donc à affecter un temps « négatif ».

Que faire si le navigateur n'accepte pas les cookies ?

- Test d'absence de cookies : if (count($_COOKIE) > 0) ...
- Trois solutions :
  - Soit passer les variables d'une page à l'autre par les formulaires.
  - Soit utiliser des variables globales.
  - Soit créer une session (voir suite du cours).

Les Cookies PHP

Un exemple

Pour déclarer un cookie :

```
$cookie_nom = "user";
$cookie_val = "Paul Durant";
setcookie($cookie_nom, $cookie_val, time() + (86400 * 30), "/"); // 86400 = 1 jour
```

Pour utiliser un cookie :

```
if(!isset($_COOKIE[$cookie_nom])) {
    echo "Cookie utilisateur '" . $cookie_nom . "' n'est pas créé!";
} else {
    echo "Cookie utilisateur  '" . $cookie_nom . "' est créé!<br>";
    echo "Sa valeur est : " . $_COOKIE[$cookie_nom];
}
?>
```

Les Sessions PHP

Qu'est-ce qu'une session?

- Une session permet de stocker des variables propres à un utilisateur ou client.
- Un identifiant et mot de passe sont typiquement des variables de session
- Les variables de session ont une durée de vie de quelques secondes à quelques heures, la durée est fixée dans la configuration initiale de PHP dans le fichier php.ini
- Les variables de sessions sont globales et sont donc visibles dans tout le site.
- A chaque session correspond un identifiant unique stockée dans l'UID (User IDentifier) de l'utilisateur ou client.

Les Sessions PHP

Débuter une session

- Avant de stocker les informations relatives à l'utilisateur, il faut lancer une session.
- La fonction session_start() lance une session, ce qui implique :
  - Enregistrement de la session du client sur le serveur.
  - Attribution d'un ID au client.

Remarque : La fonction session_start() doit être appelée **avant** la balise <html>

Les Sessions PHP

Créer et stocker des variables session

- Les variables sessions peuvent être stockées et récupérées à l'aide de la variable PHP $_SESSION :

```
session_start();
// stockage d'une variable session
$_SESSION['visitepage']=1;
```

- Exemple d'utilisation d'une variable session, création d'un compteur de visite d'un site :

```
session_start();
if (isset($_SESSION['visitepage']))
  $_SESSION['vistepage']=$_SESSION['visitepage']+1;
else
  $_SESSION['visitepage']=1;
echo "Nombre de pages visitées= ".$_SESSION['visitepage'];
```

Les Sessions PHP

Détruire une session

- La destruction d'une session implique sa destruction ainsi que la suppression de toutes les variables qui lui sont associées
- La fonction permettant de détruire une session est session_destroy().

Détruire une variable session

- La fonction unset() permet de supprimer une variable session.
- Dans l'exemple ci-dessus, pour supprimer la variable « visitepage » il suffit de faire :

```
unset (($_SESSION['visitepage']);
```

Envoi de mails en PHP

- PHP permet d'envoyer des mails directement à partir d'un script.
- La fonction mail() remplit cette tâche, sa syntaxe d'appel est la suivante :

mail(destinataire, sujet, message, entête, paramètres)

- Avec :

Paramètre	Description
destinataire	Adresse email du destinataire
sujet	Sujet du mail.
message	C'est le texte contenu dans l'email.
entête	Ce paramètre est optionnel; il spécifie des entêtes additionnels comme Cc (copie) ou Bcc (copie cachée).
paramètres	Ce paramètre est optionnel; il spécifie des paramètres additionnels pour le programme d'envoi de mail.

Envoi de mails en PHP

- Pour que la fonction mail() soit opérationnelle il faut que le serveur de mail associé à votre site soit en état de marche et autorise l'envoi de mail.
- Pour plus de détails vous pouvez vous référer aux liens :
  - http://php.net/manual/fr/function.mail.php
  - http://www.w3schools.com/php/php_ref_mail.asp

Envoi de mails en PHP : exemples

Un mail simple

```
$destinataire = "tony@unice.fr";
$sujet = "Mail test";
$message = "Bonjour, voici un message simple. ";
$auteur = paul.duraud@orange.fr;
$entete = "De: $auteur";
mail($destinataire, $sujet, $message, $entete);
echo "Mail envoyé.";
```

Envoi de mails en PHP : exemples

Un formulaire pour envoyer un mail

```
if (isset($_REQUEST['email'])) {
        //envoi de l'email
        $email = $_REQUEST['email'] ;
        $sujet = $_REQUEST['sujet'] ;
        $message = $_REQUEST['message'] ;
        mail( "tony@unice.fr", "Sujet: $sujet", $message, "De: $email" );
        echo "Merci d'utiliser notre formulaire d'email";
} else {
         echo "<form method='post' action='mailform.php'>
                Email: <input name='email' type='text' /><br />
                Sujet: <input name='sujet' type='text' /><br />
                Message:<br />
                <textarea name='message' rows='15' cols='40'></textarea><br />
                <input type='submit' value='Valider'/>
}
```

Envoi de mails en PHP : exemples

Explications relatives au code précédent :

- D'abord nous vérifions si les champs du formulaire sont bien remplis.
- Si ces champs ne sont pas remplis (par exemple lorsque la page est visitée pour la première fois), générer le formulaire HTML.
- Si les champs sont bien remplis, envoyer le mail par le formulaire.
- Après avoir rempli le formulaire, lorsque le bouton « Valider » est actionné, envoyer l'email.
- Problème : un tel envoi de mail **n'est pas sécurisé.**

Envoi de mails en PHP : exemples

- Pourquoi l'envoi n'est pas sécurisé?
- Que se passera-t-il si dans le champs input portant le nom « email » la chaine de caractère suivante est inséré?
  personne1@unice.com%0ACc:personne2@unice.com%0ABcc:person3@unice.com,personne3@unice.com,anotherpersonne4@unice.com,personne5@unice.com%0ABTo:personne6@unice.com
- Des mails sont envoyés à tous les destinataires ci-dessus, alors qu'il ne devait être destiné qu'à « tony@unice.fr »

Envoi de mails en PHP : exemples

Nous pouvons stopper les ajouts indésirables dans un email en créant la fonction suivante :

```
function spamcheck($champ_mail) {
    // utilisation de la fonction de filtrage
    $fchamp_mail=filter_var($champ_mail, FILTER_SANITIZE_EMAIL);
    // utilisation de la fonction de filtrage pour validation
    if (filter_var($fchamp_mail, FILTER_VALIDATE_EMAIL)) {
        return TRUE;
    } else {
        return FALSE;
    } }
```

- L'appel de filter_var() avec la variable **FILTER_SANITIZE_EMAIL** efface les caractères indésirables contenus dans l'adresse email
- L'appel de filter_var() avec la variable **FILTER_VALIDATE_EMAIL** valide la chaine de caractère d'un email.

Envoi de mails en PHP : exemples

Il faut donc tester le champs email avant de procéder à l'envoi comme suit :

```
if (isset($_REQUEST['email'])) {
    //vérifier si l'adresse email est valide ou non
    if ($mailcheck==FALSE) {
        echo "Entrée invalide";
    } else {
        // Voir le code d'envoi de mail ci-dessus ...
} else {
    // Générer le formulaire, voir code de génération
    // du formulaire de mail ci-dessus ...
}
```

Les filtres en PHP

Les filtres permettent de valider et filtrer les données en provenance d'une source douteuse.

Que-est-ce un filtre PHP?

Tester, valider et filtrer une donnée en provenance d'un champ d'entrée (<input .. />) où une donnée personnalisée représente une tâche importante d'une application web.

Pourquoi utiliser un filtre?

L'utilisation des filtres nous permet de nous assurer que notre application recevra en entrée des données d'un type correct.

Une règle simple : vous devez toujours filtrer des données externes !

Les filtres en PHP

Mais alors qu'entend-t-on par données externes ?

- Les données en provenance des balises input ou textarea,
- Les cookies,
- Les données des services Web,
- Les variables du serveur,
- Les résultats d'une requête dans une base de données.

Les filtres en PHP

Les fonctions de filtrage

- filter_var() : filtre une seule variable avec un filtre spécifique,
- filter_var_array() : filtre plusieurs variables avec le même filtre ou avec différents filtres,
- filter_input() : récupère une variable d'une balise « input » et la filtre,
- filter_input_array() : récupère plusieurs variables à partir de balises « input » et les filtrent avec le même filtre ou avec des filtres différents.

La liste complète des fonctions de filtrage est donnée dans :

- php.net/manual/fr/ref.filter.php
- http://www.w3school.com/php/php_ref_filter.asp

Les filtres en PHP

Exemple ...

Validation d'un entier en utilisant la fonction filter_var() :

```
$int = 123;

if(!filter_var($int, FILTER_VALIDATE_INT)) {
    echo("La variable n'est pas un entier");
} else {
    echo("La variable est un entier");
}
```

Les filtres en PHP

Filtres pour valider

- Utilisés pour valider des données en provenance de balises <input> ou <textarea>.
- Utilisent des règles de formatages strictes (comme les règles pour valider les URL ou e-mail).
- Renvoient le type attendu si validation et FALSE si rejet.

Les filtres en PHP

Filtres pour assainir

- Ils sont utilisés pour accepter ou refuser des caractères spécifiques dans une chaine de caractère.
- Ils n'utilisent pas de règles formatées.
- Ils retournent toujours une chaine de caractère.

Les filtres en PHP

Les options et flags

- Ils sont utilisés pour ajouter des options de filtrages additionnelles à des filtres spécifiques.
- Des filtres différents ont des options et des flags différents.
- Les options peuvent être mises dans une table associative portant le nom « options ». Si un seul flag est utilisé, il n'est pas nécessaire de le mettre dans une table.

Les filtres en PHP

Exemple...

Nous validons un entier en utilisant la fonction filter_var() et les options « min_range » (valeur minimale) et « max_range » (valeur maximale) :

```
$var=300;
$int_options = array(
            "options"=>array(
                                    "min_range"=>0,
                                    "max_range"=>256 ))

if (!filter_var($var, FILTER_VALIDATE_INT, $int_options)) {
      echo("La variable n'est pas valide");
} else {
      echo("La variable est valide");
}
```

Les filtres en PHP

Valider une entrée

Nous validons une donnée entrée par un formulaire <form> :

- La première vérification porte sur l'existence de la donnée avec la fonction **filter_has_var()**.
- La seconde consiste à filtrer la donnée en utilisant la fonction **filter_input().**

Les filtres en PHP

Exemple...

Testons et filtrons une variable « email » transmise à partir d'un formulaire avec la méthode « get » :

```
if(!filter_has_var(INPUT_GET, "email")) {
    echo("La variable en entrée 'email' n'existe pas");
} else {
    if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL)) {
        echo "L'e-mail est invalide";
    } else {
        echo "L'e-mail est valide";
    }
}
```

Les filtres en PHP

Assainir une entrée

Nous allons maintenant assainir une entrée en filtrant les caractères indésirables qu'elle peut contenir :

```
if(!filter_has_var(INPUT_POST, "url")) {
    echo("La variable en entrée 'url' n'existe pas ");
} else {
    $url = filter_input(INPUT_POST, "url",
FILTER_SANITIZE_URL);
}
```

Donnera pour www.àunicée.fr => www.unice.fr

Les filtres en PHP

Filtrer et assainir plusieurs entrées

- Dans la plupart des cas nous sommes amenés à filtrer plus d'une entrée.
- Pour éviter des appels successifs à la fonction filter_var() ou filter_input(), nous utilisons :
  - filter_var_array(),
  - filter_input_array().
- Nous sommes donc amenés à définir une table associative adaptée aux contraintes de filtrage.

Les filtres en PHP

Exemple...

```
$filters = array(    "nom" => array("filter"=>FILTER_SANITIZE_STRING ),
                     "age" => array("filter"=>FILTER_VALIDATE_INT,
                                    "options"=>array("min_range"=>1,
                                                     "max_range"=>120)),
                     "email"=> FILTER_VALIDATE_EMAIL,
                     );

$result = filter_input_array(INPUT_GET, $filters);
```

On remarque que la constante passée à filter_input_array est INPUT_GET car le formulaire est généré par l'attribut « method » affecté à « get ».

Les filtres en PHP

Exemple... Suite

```
if (!$result["age"]) {
    echo("L'age doit se situer entre 1 et 120.<br />");
} elseif(!$result["email"]) {
    echo("L'e-mail est invalide.<br />");
} else {
    echo("Les entrées sont valides.");
}
```

La table de booléens $result est retournée par la fonction dont les coordonnées sont les noms des variables à filtrer. Si le filtrage indique que la valeur de la variable est correcte, $result[« nom_de_la_varaible »] contient TRUE sinon FALSE.

Les filtres en PHP

L'exemple précédent permet donc d'observer que :

- filter_input_array() comporte une seconde variable d'entrée qui peut être une table ou un identifiant de filtrage.
- si le second paramètre est celui d'un identifiant toutes les valeurs d'entrées sont filtrées de la même manière.

Si le second paramètre est une table, elle est soumise aux règles suivantes :

- La table doit être associative, ses clefs doivent être constituées par les noms des variables à filtrer.
- La valeur des éléments de la table sont les identifiants de filtrage ou des tables associatives.
- Dans le cas des tables associatives :
  - La première clef de la table est « filter »,
  - Si le filtrage requiert des options, la seconde clef est « options » la valeur associée à cette clef est une table contenant les valeurs des options.

Les filtres en PHP

Utilisation de filtres sur des variables courantes : option FILTER_CALLBACK

- Il est possible de filtrer des variables courantes.
- Les paramètres du filtre étant passés de la même manière que précédemment.

Les filtres en PHP

Exemple...

L'exemple qui suit indique comment procéder pour transformer tous les « _ » d'une chaine de caractère en espace.

```
function convertSpace($string) {
    return str_replace("_", " ", $string);
}

$string = "Paul_est_un_adolescent";

echo filter_var($string, FILTER_CALLBACK,
                        array("options"=>"convertSpace"));
```

La fonction précédente retourne : Paul est un adolescent

Les filtres en PHP

Dans l'exemple précédent nous avons vu que tous les « _ » ont été convertis en espace en deux étapes :

- Une fonction converSpace() qui remplace les « _ » en espace est créée.
- La fonction filter_var() est appelée pour filtrer la chaine $string. Outre la variable $string à filtrer, nous passons par paramètre à filter_var(), l'identifiant « FILTER_CALLBACK » et une table indiquant que la fonction de filtrage à utiliser est convertSpace().

Les fonctions diverses de PHP

Dans ce chapitre nous allons étudier les fonctions dites « diverses » qui ne sont classées dans aucun des thèmes précédents mais qui doivent être connues.

- int **connection_aborted** (void) : Indique si l'internaute a abandonné la connexion HTTP.
- int **connection_status** (void) : Retourne les bits de statut de la connexion HTTP.
- mixed **constant** (string $name) — Retourne la valeur d'une constante

Les fonctions diverses de PHP

- bool **defined** (string $name) : Vérifie l'existence d'une constante.
- die — Alias de la fonction *exit.*
- void **exit** ([string $status]) : Affiche un message et termine le script courant.
- mixed **eval** (string $code) : Exécute une chaîne comme un script PHP. Attention l'utilisation de cette fonction est vivement déconseillée.
- mixed **get_browser** ([string $user_agent [, bool $return_array = false]]) : Indique les capacités du navigateur client

Les fonctions diverses de PHP

- void **__halt_compiler** (void) : Stoppe l'exécution du compilateur.
- mixed **highlight_file** (string $filename [, bool $return = false]) : Colorisation syntaxique d'un fichier.
- mixed **highlight_string** (string $str [, bool $return = false]) : Applique la syntaxe colorisée à du code PHP.
- int **ignore_user_abort** ([string $value]) : lors de la déconnexion du client Web, le script poursuit son exécution.
- string **pack** (string $format [, mixed $args [, mixed $...]]) : Compacte des données dans une chaîne binaire.

Les fonctions diverses de PHP

- bool **php_check_syntax** (string $filename [, string &$error_message]) : Vérifie la syntaxe PHP d'un fichier spécifique et l'exécute.
- string **php_strip_whitespace** (string $filename) : Retourne la source sans commentaires, ni espaces blancs.
- show_source : Alias de highlight_file()
- int **sleep** (int $seconds) : Arrête l'exécution durant quelques secondes.
- array **sys_getloadavg** (void) : Récupère la charge moyenne du système

Les fonctions diverses de PHP

- mixed **time_nanosleep** (int $seconds , int $nanoseconds) : Attendre pendant un nombre de secondes et de nanosecondes
- tbool **time_sleep_until** (float $timestamp) : Arrête le script pendant une durée spécifiée
- string **uniqid** ([string $prefix = "" [, bool $more_entropy = false]]): Génère un identifiant unique.
- array **unpack** (string $format , string $data) : Déconditionne des données depuis une chaîne binaire
- void **unsleep** (int $micro_seconds) : Arrête l'exécution durant quelques microsecondes

Traitements simples de fichiers XML

- Un fichier XML permet de stocker, transférer et afficher simplement des informations.
- Un fichier XML a une structure lisible par la plupart des langages et par tous les navigateurs.
- XML est un langage comportant des balises comme XHTML :
  - Un fichier XML est un fichier texte organisé sous forme arborescente.
  - XML n'a pas de balises prédéfinies, c'est le programmeur qui les crée.
- XML est conçu pour décrire son contenu par lui même

Traitements simples de fichiers XML

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
        <librairie>
                <livre categorie="Langue">
                        <titre langue="fr">Apprendre Italien</titre>
                        <auteur>Simone Ziani</auteur>
                        <annee>2005</annee>
                        <prix>30.00</prix>
                </livre>
                <livre categorie="Enfant">
                        <titre langue="fr">Harry Potter</titre>
                        <auteur>J K. Rowling</auteur>
                        <annee>2005</annee>
                        <prix>22.99</prix>
                </livre>
                <livre categorie="Web">
                        <titre langue="en">Learning XML</titre>
                        <auteur>Erik T. Ray</auteur>
                        <annee>2008</annee>
                        <prix>20.99</prix>
                </livre>
        </librairie>
```

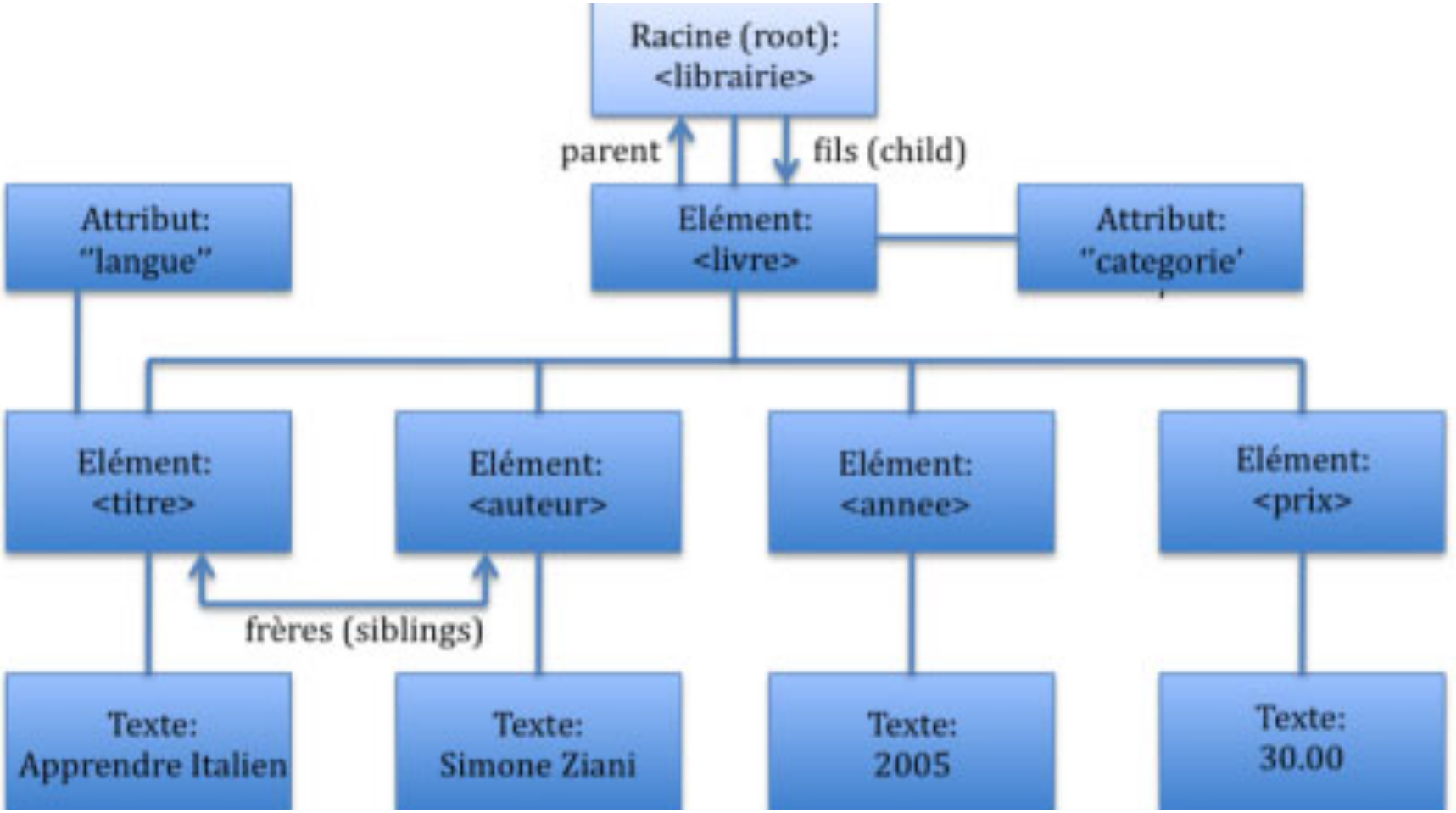
Traitements simples de fichiers XML

Quelques définitions :

- Dans l'exemple précédent les balises forment les **éléments** du fichier, par exemple <librarie> .. </librairie>
- La balise <livre> est un élément **fils** de l'élément <librairie>
- « categorie », « langue » sont des **attributs** de des éléments <livre> et <titre>

Traitements simples de fichiers XML

Schéma de l'arborescence



Traitements simples de fichiers XML

- Nous allons voir dans cette section les outils PHP permettant de « parser » les éléments d'un fichier XML.
- La classe SimpleXMLElement s'identifie un élément du document XML, les méthodes qui y sont rattachées sont :
  - SimpleXMLElement::addAttribute — Ajoute un attribut à l'élément SimpleXML
  - SimpleXMLElement::addChild — Ajoute un élément enfant au noeud XML
  - SimpleXMLElement::asXML — Retourne une chaîne XML basée sur un élément SimpleXML
  - SimpleXMLElement::attributes — Identifie les attributs d'un élément
  - SimpleXMLElement::children — Cherche les fils d'un noeud donné
  - SimpleXMLElement::__construct — Crée un nouvel objet SimpleXMLElement
  - SimpleXMLElement::count — Compte le nombre de fils pour un élément

Traitements simples de fichiers XML

- Méthode rattachées à la classe SimpleXMLElement (suite) :
  - SimpleXMLElement::getDocNamespaces — Retourne les espaces de noms déclarés dans un document
  - SimpleXMLElement::getName — Récupère le nom d'un élément XML
  - SimpleXMLElement::getNamespaces — Retourne les espaces de noms utilisés dans un document
  - SimpleXMLElement::registerXPathNamespace — Crée un contexte préfixe/ns pour la prochaine requête Xpath
  - SimpleXMLElement::saveXML — Alias de SimpleXMLElement::asXML
  - SimpleXMLElement::__toString — Retourne le contenu sous forme de chaine
  - SimpleXMLElement::xpath — Exécute une requête Xpath sur des données XML

Traitements simples de fichiers XML

- La classe SimpleXMLIterator fournit des itérations récursives sur tous les éléments d'un objet :
  - SimpleXMLIterator::current — Retourne l'entrée courante
  - SimpleXMLIterator::getChildren — Retourne un itérateur pour l'entrée courante, si c'est un objet SimpleXML
  - SimpleXMLIterator::hasChildren — Indique si l'entrée courante de SimpleXML est un objet
  - SimpleXMLIterator::key — Retourne la clé courante
  - SimpleXMLIterator::next — Se déplace sur l'entrée SimpleXML suivante
  - SimpleXMLIterator::rewind — Replace le pointeur SimpleXML au début
  - SimpleXMLIterator::valid — Vérifie si une ressource SimpleXML contient d'autres entrées

Traitements simples de fichiers XML

Les fonctions SimpleXML

- simplexml_import_dom — Construit un objet SimpleXMLElement à partir d'un objet DOM
- simplexml_load_file — Convertit un fichier XML en objet
- simplexml_load_string — Convertit une chaîne XML en objet

Traitements simples de fichiers XML

Exemples...

```
<?php
      $dom = new DOMDocument;
      $dom->loadXML('<books><book><title>Harry Potter</title></
      book></books>');
      if (!$dom) {
       echo 'Erreur durant l\'analyse du document';
      exit;
      }

      $s = simplexml_import_dom($dom);

      echo $s->book[0]->title;
?>
```

L'exécution de ce code donne : Harry Potter

Traitements simples de fichiers XML

Exemples...

```
<?php
	libxml_use_internal_errors(true);
	$myXMLData =
		"<?xml version='1.0' encoding='UTF-8'?>
			<document>
				<user>John Doe</wronguser>
				<email>john@example.com</wrongemail>
			</document>";

	$xml = simplexml_load_string($myXMLData);
	if ($xml === false) {
		 echo "Failed loading XML: ";
		foreach(libxml_get_errors() as $error) {
			echo "<br>", $error->message;
		}
	} else {
		print_r($xml);
	}
?>
```

L'exécution de ce code donne :

```
Failed loading XML:
Opening and ending tag mismatch: user line 3 and wronguser
Opening and ending tag mismatch: email line 4 and wrongemail
```

Traitements simples de fichiers XML

Exemples...

```
<?php
	$myXMLData =
		"<?xml version='1.0' encoding='UTF-8'?>
		<note>
			<to>Tove</to>
			<from>Jani</from>
			<heading>Reminder</heading>
			<body>Don't forget me this weekend!</body>
		</note>";

	$xml=simplexml_load_string($myXMLData) or die("Error: Cannot create object");
	print_r($xml);
?>
```

L'exécution de ce code donne :

SimpleXMLElement Object ([to] => Tove [from] => Jani [heading] => Reminder [body] => Don't forget me this weekend!)

PHP et les SGBD

- PHP permet de créer une interface entre un serveur WEB et un système de gestion de base de données (SGBD).
- Plusieurs SGBD comme ORAQLE, SQL/Server, SYBASE, PostgreSQL, MySQL ont développés des interfaces pour PHP.
- Les principes de fonctionnement de ces interfaces sont similaires.
- Par la suite nous étudierons l'interface MySQL.

PHP et MySQL : Introduction

- MySQL est un SGBD permettant de stocker des bases de données constituées elles mêmes de tables.
- Les bases de données sont utiles lorsque l'on veut stocker des informations par catégorie.
- Une société par exemple, aura des tables suivant les catégories « Employés », « Produits », « Clients », « Commandes »…
- Vous pouvez télécharger gratuitement un serveur PHP avec une base de données MySQL à l'URL : http://www.mysql.com/downloads/index.html
- Vous pouvez aussi simuler une application web complète sur votre PC soient **PHP** + **Apache** + **MySQL** + **PhpMyAdmin** en téléchargeant http://www.easyphp.org/

PHP et MySQL : Les Tables

- Une base de données contient en général une ou plusieurs tables.
- Chaque table est identifiée par un nom comme « Clients » ou « Commandes ».
- Exemple :

Nom	Prénom	Adresse	Ville
Durant	Pierre	10 rue de Paris, 06000	Nice
Nicolini	Evelyne	20b bd. De Montreal, 06200	Nice
Astier	Paul	37/41 bd. Dubouchage, 06000	Nice

PHP et MySQL : Les Requêtes

- Une requête est une question ou une demande.
- Formuler une requête à une base de données permet d'obtenir une information spécifique et avoir en retour un ensemble d'enregistrements correspondant à la demande.
- Exemple, la requête :

  SELECT Nom FROM Personne

  appliquée à la table précédente donne :

Nom
Durant
Nicolini
Astier

PHP et MySQL : Connexion

- Pour se connecter sur un SGBD nous avons besoin :
  - D'un nom de serveur,
  - D'un nom d'utilisateur,
  - D'un mot de passe.
- Dans le cas de MySQL la syntaxe de la fonction de connexion est :

  mysql_connect(nom_du_serveur,nom_utilisateur,mot_de_passe);
- **Remarque :** Il y a plus de paramètres d'entrées possibles pour cette fonction.
  Pour plus d'informations nous pouvons consulter les liens :
  - http://php.net/manual/en/book.mysql.php
  - http://www.w3schools.com/php/php_ref_mysql.asp

PHP et MySQL : Connexion

- Exemple :

```
<?php
        $connect = mysql_connect("localhost","paul","abc123");

        if (!$connect) {
                die("Connexion impossible:"       . mysql_error());
        }

        // lignes de code …
?>
```

- Dans ce cas nous nous connectons sur le serveur local mais le SGBD peut se trouver sur un autre serveur dont il faut alors préciser le numéro IP ou l'adresse. La valeur par défaut est « localhost:3306 »
- Dans le cas d'absence de mot de passe il suffit de remplacer "abc123" par ""

PHP et MySQL : Connexion

- La connexion se ferme automatiquement lorsque le script se termine.
- Pour fermer une connexion avant la fin du script il faut utiliser la fonction **mysql_close**() :

```
<?php
    $connect = mysql_connect("localhost","paul","abc123");
    if (!$connect) {
        die('Connexion impossible: ' . mysql_error());
    }
    // lignes de code …

    mysql_close($connect);
?>
```

PHP et MySQL : Base de données

- L'instruction **CREATE DATABASE** est utilisée pour créer une base de données :

  CREATE DATABASE nom_base_de_donnee

- Les liens qui suivent vous offrent une documentation complète sur le langage SQL :
  - http://www.sqltutorial.org/
  - http://www.w3schools.com/sql/default.asp
- La fonction mysql_query() permet d'exécuter une requête SQL ou envoyer une commande à MySQL :

```
if (mysql_query("CREATE DATABASE my_db",$connect)) {
        echo "Database created";
} else {
        echo "Error creating database: " . mysql_error();
}
mysql_close($connect);
```

PHP et MySQL : Tables

- L'instruction CREATE TABLE est utilisée pour créer une table .
- La syntaxe de cette requête SQL est la suivante :

```
CREATE TABLE nom_table (
        nom_colonne1    type_donnee,
        nom_colonne2    type_donnee,
        nom_colonne3    type_donnee
)
```

- Exemple :

```
mysql_select_db("my_db", $connect);
$sql = "CREATE TABLE Personne (
        Nom varchar(15),
        Prenom varchar(15),
        Age int)";

// Execution de la requête
mysql_query($sql,$connect);

mysql_close($connect);
```

PHP et MySQL : Tables

- **Attention :** Une base de données doit être sélectionnée par la fonction **mysql_select_db**() avant la création d'une table.

- **Remarque :** Lorsqu'un champ de type « varchar » (soit une chaîne de caractère) est créé dans une table, la taille maximale de la chaîne doit être spécifiée, soit varchar(15) dans notre exemple.

- Le type permet de préciser le type de donnée que la colonne peut stocker. Pour connaître l'ensemble des types possibles vous pouvez vous référer aux liens :
  - http://dev.mysql.com/doc/refman/5.0/en/data-types.html,
  - http://www.w3schools.com/sql/sql_datatypes.asp

PHP et MySQL : Clés Primaires et Champs auto-incrémentés

- Une clef primaire permet d'identifier de manière unique la ligne d'une table.
- Le champ d'une clé primaire ne peut être nul; lors de la création du champ l'option « **NOT NULL** » doit être utilisée.
- Très souvent, une clef primaire est un numéro d'identifiant unique qui est couplé à l'instruction **AUTO_INCREMENT**.
- AUTO_INCREMENT incrémente de un la valeur de la clé primaire à chaque nouvel enregistrement

PHP et MySQL : Clés Primaires et Champs auto-incrémentés

- Exemple, la création d'une table « Personne » contenant une clef primaire :

```
$sql = "CREATE TABLE Personne (
    ID_personne int NOT NULL AUTO_INCREMENT,
    PRIMARY KEY(ID_personne),
    Nom varchar(15),
    Prenom varchar(15),
    Age int)";

mysql_query($sql,$connect);
```

PHP et MySQL : Insertion dans une table

- L'instruction SQL « INSERT INTO » est utilisée pour insérer des données dans une table.
- La syntaxe de cette instruction peut être exprimée de deux manières :
  - La première ne spécifie pas les noms des colonnes où doivent être insérées les données :

    INSERT INTO nom_table
    VALUES (valeur1, valeur2, valeur3,...)
  - La seconde spécifie à la fois les noms des colonnes et les valeurs à insérer :

    INSERT INTO nom_table (colonne1, colonne2, colonne3,...)
    VALUES (valeur1, valeur2, valeur3,...)

PHP et MySQL : Insertion dans une table

- Exemple, nous avions créé une table « Personne » qui contenait trois colonnes « Nom », « Prenom » et « Age », nous allons maintenant insérer deux nouveaux enregistrements dans la table « Personne » :

```
mysql_query("INSERT INTO Personne (Prenom, Nom, Age)
                    VALUES ('Pierre', 'Le Grand', '35') », $connect);

mysql_query("INSERT INTO Personne (Prenom, Nom, Age)
                    VALUES ('Gérard', 'Simard', '33') », $connect);

mysql_close($connect);
```

PHP et MySQL : Insertion multiple dans une table

- Pour enchaîner plusieurs requêtes d'insertions nous faisons appel à la fonction **mysqli_multi_query()**.
- Exemple :

```
$sql = "INSERT INTO Personne (prenom, nom, age)
        VALUES ('John', 'Doe', '24');";
$sql .= "INSERT INTO Personne (prenom, nom, age)
        VALUES ('Mary', 'Moe', '20');";
$sql .= "INSERT INTO Personne (prenom, nom, age)
        VALUES ('Julie', 'Dooley', '22')";

if (mysqli_multi_query($conn, $sql)) {
        echo "Les nouvelles insertions ont été exécutées";
} else {
        echo "Erreur d'insertion : " . $sql . "<br>" .
mysqli_error($conn);
}
```

PHP et MySQL : Insertion avec auto-incrémentation de clé primaire

- Considérons une table avec dont le champ id est une clé primaire auto incrémentée,
- Chaque insertion implique l'auto incrémentation automatique du champ id.
- La fonction mysqli_insert_id permet de récupérer la dernière valeur du champ id.

PHP et MySQL : Insertion avec auto-incrémentation de clé primaire

- Exemple: Soit la table :

```
CREATE TABLE Personne (
        id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        nom VARCHAR(30) NOT NULL,
        prenom VARCHAR(30) NOT NULL,
        age INT(3),
        date_saisie TIMESTAMP)
```

- Le code pour insérer un élément et récupérer ensuite son Id est :

```
$sql = "INSERT INTO Personne (prenom, nom, age)
VALUES ('John', 'Doe', '23')";

if (mysqli_query($conn, $sql)) {
        $last_id = mysqli_insert_id($conn);
        echo ”Nouvel enregistrement ok, le nouvel ID est : " . $last_id;
} else {
        echo "Erreur: " . $sql . "<br>" . mysqli_error($conn);
}
```

PHP et MySQL : Insertion de données dans une table par un formulaire

- Nous allons maintenant créer un formulaire HTML qui nous permette d'insérer des enregistrements dans la table Personne.

- Soit le formulaire :

```
<form action="inserer.php" method="post">
        Prénom: <input type="text" name="prenom" />
        Nom: <input type="text" name="nom" />
        Age: <input type="text" name="age" />
        <input type="submit" value="Valider"/>
 </form>
```

PHP et MySQL : Insertion de données dans une table par un formulaire

- Le code PHP contenu dans le fichier « inserer.php » est le suivant :

```
mysql_select_db("my_db", $connect);

$sql=   "INSERT INTO Personne (Nom, Prenom, Age)
         VALUES
         ('".$_POST["nom"]."','".$_POST["prenom"]."','".$_POST["age"]."')";
if (!mysql_query($sql,$connect)) {
     die('Erreur: ' . mysql_error());
echo "un enregistrement a été ajouté";

mysql_close($connect)
```

PHP et MySQL : Sélection de données dans une table

- L'instruction SQL « SELECT » permet de sélectionner des données dans une table :

```
SELECT nom_colonne(s)
FROM nom_table
```

- Exemple :

```
mysql_select_db("my_db", $connect);

$result = mysql_query("SELECT * FROM Personne", $connect);

while($row = mysql_fetch_array($result)) {
    echo $row['Nom'] . " " . $row['Prenom'];
    echo "<br />";
}
mysql_close($connect);
```

- La fonction **mysql_fetch_array**() permet à chaque appel de récupérer une ligne de données correspondant à celles sélectionnées dans la table.

PHP et MySQL : Sélection de données dans une table avec filtrage

- L'instruction SQL « WHERE » permet d'extraire des données qui remplissent certains critères.
- La syntaxe de l'instruction WHERE s'écrit :

```
SELECT nom_colonne(s)
FROM nom_table
WHERE nom_colonne operator valeur
```

PHP et MySQL : Sélection de données dans une table avec filtrage

- L'exemple qui suit permet de sélectionner dans la table Personne toutes les lignes où le champ « Prenom » a la valeur « Pierre ».

```
mysql_select_db("my_db", $connect);
$result = mysql_query("SELECT * FROM Personne
                       WHERE Prenom='Pierre'", $connect);
while($row = mysql_fetch_array($result)) {
    echo $row['Nom'] . " " . $row['Prenom'];
    echo "<br />";
}
```

- L'exécution du code précédent affiche :

```
Le Grand Pierre
```

PHP et MySQL : Sélection de données dans une table avec filtrage

- L'instruction SQL « ORDER BY » complétée d'un mot clef permet de trier les données parmi celles qui ont été sélectionnées.
- La syntaxe de l'instruction SQL devient alors :

```
SELECT nom_colonne(s)
FROM nom_table
ORDER BY nom_colonne(s) ASC|DESC
```

- Les mots clés ASC et DESC permettent un classement ascendant ou descendant. Par défaut, c'est le mot clé DESC qui est pris.

PHP et MySQL : Sélection de données dans une table avec filtrage

- Exemple, sélection de toutes les personnes enregistrées dans la table Personne, triées d'après leur âge :

```
mysql_select_db("my_db", $connect);
$result = mysql_query("SELECT * FROM Personne ORDER BY age", $connect);
while($row = mysql_fetch_array($result)) {
    echo $row['Nom'];
    echo " " . $row['Prenom'];
    echo " " . $row['Age'];
    echo "<br />";
}
mysql_close($connect);
```

- Le résultat de l'exécution de ce code affiche :

```
Simard Gérard 33
Le Grand Pierre 35
```

PHP et MySQL : Sélection de données dans une table avec filtrage

- L'instruction Order By peut être appliquée à deux colonnes d'une table
- Lorsque l'on tri sur plus d'une colonne, la seconde colonne est utilisée lorsque les valeurs de la première colonne sont égales.
- La syntaxe de l'instruction est la suivante :

```
SELECT nom_colonne(s)
FROM nom_table
ORDER BY colonne1, colonne2
```

PHP et MySQL : ORDER BY

Exemple soit la table « Personne » dont le contenu est :

Nom	Prenom	AnneeNaissance
Thomas	Alva Edison	1847
Benjamin	Franklin	1706
Thomas	More	1478
Thomas	Jefferson	1826

La requête : SELECT * FROM Personne ORDER BY Nom DESC, AnneeNaissance ASC

Donne :

Nom	Prenom	AnneeNaissance
Thomas	More	1478
Thomas	Jefferson	1826
Thomas	Alva Edison	1847
Benjamin	Franklin	1706

PHP et MySQL : Sélection de données dans une table avec filtrage et limitations

- Dans le cas où l'on a un grand nombre de réponses et que l'on veut faire une « pagination » de l'affichage, nous pouvons imposer des limites dans notre requête SQL comme dans l'exemple ci-dessous :
  $sql = "SELECT * FROM Personne **LIMIT 30**";
- Cette requête limitera la sélection de la 1ière à la 30ième donnée de la table « Personne ».
- Dans le cas où l'on veut « paginer » c'est à dire de sélectionner par tranche notre requête deviendra :
  $sql = "SELECT * FROM Personne **LIMIT 10 OFFSET 15**" ;
  ou $sql = "SELECT * FROM Personne **LIMIT 15, 10**";
- Cette requête impliquera la sélection entre la 16ème et 25ème données de la table « Personne ».

PHP et MySQL : Mise à jour de données dans une table

- L'instruction « UPDATE » est utilisée pour mettre à jour des données dans une table.

- Sa syntaxe est la suivante :

  UPDATE nom_table
  SET colonne1=valeur1, colonne2=valeur2,...
  WHERE colonnex=valeurx

- **Remarque :** La clause WHERE est utilisée pour spécifier quelles données de la table doivent être mise à jour, l'absence de l'utilisation de la clause WHERE implique la mise à jour de toute la table !

PHP et MySQL : Mise à jour de données dans une table

- Exemple, nous voulons mettre à jour l'âge d'une personne enregistrée dans la table Personne :

```
mysql_select_db("my_db", $connect);
mysql_query("UPDATE Personne SET Age = '36'
    WHERE   Prenom = 'Pierre' AND
                Nom = 'Le Grand'", $connect);
mysql_close($connect);
```

- L'exécution de ce code va affecter à 36 l'âge de la personne appelée Pierre Le Grand.

PHP et MySQL : Suppression de données dans une table

- L'instruction SQL « DELETE FROM » permet d'effacer les enregistrements d'une table.
- La syntaxe de cette instruction s'écrit :

  DELETE FROM nom_table
  WHERE colonnex = valeurx

- **Remarque :** Comme dans les cas précédents, la clause WHERE complète l'instruction DELETE FROM pour préciser quels enregistrements doivent être effacés. L'omission de l'utilisation de la clause WHERE, implique l'effacement de tous les enregistrements de la table !

PHP et MySQL : Suppression de données dans une table

- L'exemple qui suit illustre l'effacement de l'enregistrement « Pierre Le Grand » de la table « Personne » :

```
mysql_select_db("my_db", $connect);

mysql_query("DELETE FROM Personne WHERE Prenom = 'Pierre' AND Nom='Le Grand'", $connect);

mysql_close($connect);
```

PHP et MySQL : Interface Orientée Objet

- Nous allons maintenant repasser en revue les fonctions d'interface de MySQL précédentes dans leur version orientée objet.
- Connexion :

```
$servername = "localhost";
$username = "paul";
$password = "abc123";

// Création de la connexion
$conn = new mysqli($servername, $username, $password);

// Vérification de la connexion
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

PHP et MySQL : Interface Orientée Objet - BD

- Création d'une base de données :

```
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Base de données créée avec succès";
} else {
    echo "Erreur dans la création de la base de données : " . $conn->error;
}
// fermeture de la connexion
$conn->close();
```

PHP et MySQL : Interface Orientée Objet - Table

- Création d'une table :

```
$sql = "CREATE TABLE Personne (
            id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
            prenom VARCHAR(30) NOT NULL,
            nom VARCHAR(30) NOT NULL,
            email VARCHAR(50),
            date_saisie TIMESTAMP
            )";

if ($conn->query($sql) === TRUE) {
      echo "Table Personne créée avec succès";
} else {
      echo "Erreur de création de la table : " . $conn->error;
}

$conn->close();
```

PHP et MySQL : Interface Orientée Objet - Insertion

- Comme nous l'avons vu précédemment, l'insertion se fait aussi par l'intermédiaire d'une requête SQL dont l'exécution s'opère par le biais de la méthode :

  $conn->query(« requête SQL »);

- Dans le cas d'une clé primaire auto incrémentée la récupération l'ID de le dernière insertion se fait par la propriété :

  $conn->insert_id

- Dans le cas d'insertions multiples la méthode est :

  $conn->multi_query(« requête SQL »);

PHP et MySQL : Interface Orientée Objet - Insertion

- L'insertion de requête prédéfinie se fait en deux étapes :
  - la **préparation** de la requête, soit par exemple :
    $sql = "INSERT INTO Personne (prenom, nom, age) VALUES (**?, ?, ?**)";
    $stmt = **$conn->prepare**($sql);
    $prenom = "Paul"; $nom = "Dupont"; $age = 22;
    $stmt->bind_param("ssi", $prenom, $nom, $age);
  - l'**exécution** :
    $prenom = "Pierre"; $nom = "Simon"; $age = 20;
    $stmt->execute();
- **Remarque :** la chaîne « ssi » apparaissant dans bind_param est définie en fonction des types des paramètres insérés à savoir :
  - s pour string,
  - i pour integer,
  - d pour double
  - b pour blob.

PHP et MySQL : Interface Orientée Objet - Sélection

- La sélection de données dans une table et leur exploitation se fait en deux étapes :
  - $sql = "SELECT id, nom, prenom FROM Personne";
    $result = $conn->query($sql);
  - L'objet $result contient les données retrouvées dans la table personne :

```
if ($result->num_rows > 0) {
        // récupération des résultats
        while($row = $result->fetch_assoc()) {
            echo "id: " . $row["id"]. " - Nom: " . $row["nom"]. "  -
Prénom " . $row["prenom"]. "<br>";
        }
} else {
        echo "Aucun résultat";
}
```

PHP et MySQL : Interface Orientée Objet – Update et Delete

- La mise à jour de données se fait de la même manière en utilisant $conn->query(« requête SQL ») :

```
$sql = "UPDATE Personne SET nom='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
    echo "Mise à jour effectuée avec succès";
} else {
    echo "Erreur dans la mise à jour: " . $conn->error;
}
```

- L'effacement d'une donnée emploie aussi $conn->query :

```
$sql = "DELETE FROM Personne WHERE id=3";

if ($conn->query($sql) === TRUE) {
    echo "Effacement effectué avec succès";
} else {
    echo "Erreur dans l'effacement : " . $conn->error;
}
```

PHP et PDO : PHP Data Objects

- L'interface orientée objet de PHP appelée PDO permet de travailler avec 12 SGBD différents.
- PDO remplit les mêmes fonctionnalités vues précédemment.
- Pour observer les détails de l'installation de PDO nous pouvons consulter l'URL : http://php.net/manual/fr/pdo.installation.php
- Nous allons passer en revue rapidement ces fonctionnalités, pour plus de détails se reporter à l'URL : http://php.net/manual/fr/book.pdo.php

PHP et PDO : Connexion et Fermeture

- Connexion :

```
try {
        $conn = new PDO("mysql:host=
 $servername;dbname=myDB",
                                    $username, $password);
        // affectation de PDO au mode d'exception d'erreur
        $conn->setAttribute(PDO::ATTR_ERRMODE,
                                    PDO::ERRMODE_EXCEPTION);
        echo "Connected successfully";
    }
catch(PDOException $e)
    {
        echo "La connexion a échoué : " . $e->getMessage();
    }
```

- Fermeture :

```
$conn = null;
```

PHP et PDO : Création d'une table

```
try {
        $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
        // affectation de PDO au mode d'exception d'erreur
         $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        // construction de la requête SQL ... Voir plus haut

        // utilisation d'exec() en l'absence de données retournées
         $conn->exec($sql);
        echo "La table Personne est créée avec succès";
    }
catch(PDOException $e)
    {
        echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
```

PHP et PDO : Insertion

```
try {
        $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
        // affectation de PDO au mode d'exception d'erreur
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        // construction de la requête SQL ... Voir plus haut

        // utilisation d'exec() en l'absence de données retournées
        $conn->exec($sql);
        echo "Enregistrement effectué avec succès";
    }
catch(PDOException $e)
    {
        echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
```

PHP et PDO : Dernier Id entré

```
try {
        $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
        // affectation de PDO au mode d'exception d'erreur
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        // construction de la requête SQL ... Voir plus haut

        // utilisation d'exec() en l'absence de données retournées
        $conn->exec($sql);
        // récupération du dernier ID
        $last_id = $conn->lastInsertId();
        echo "L'enregistrement est effecuté, le dernier ID est : " . $last_id;
    }
catch(PDOException $e)
    {
         echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
```

PHP et PDO : Requête prédéfinie

```
try {
        $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
        // affectation de PDO au mode d'exception d'erreur
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        // préparation de la requête
        $stmt = $conn->prepare("INSERT INTO Personne (nom, prenom, age)
                                VALUES (:nom, :prenom, :age)");
        $stmt->bindParam(':nom', $nom);
        $stmt->bindParam(':prenom', $prenom);
        $stmt->bindParam(':age, $age);

         // insertion d'une ligne
        $prenom = "John"; $nom = "Doe"; $age = 25;
        $stmt->execute();

        // insertion d'une nouvelle ligne
        $prenom = "Julie"; $nom = "Dooley"; $age = 22;
        $stmt->execute();

        echo "Les nouveaux enregistrements ont été effectués";
   }
catch(PDOException $e)
   {
        echo "Error: " . $e->getMessage();
   }
$conn = null;
```

PHP et PDO : Sélection

```
try {
        $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
        // affectation de PDO au mode d'exception d'erreur
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);


        // Requête
        $sql = "SELECT id, prenom, nom FROM Personne";
        $result = $conn->query($sql);

        if ($result->num_rows > 0) {
                echo "<table><tr><th>ID</th><th>Name</th></tr>";
                // a chaque ligne de donnée trouvée construire la table
                while($row = $result->fetch_assoc()) {
                        echo "<tr><td>".$row["id"]."</td><td>".$row[”prenom"]." ".$row["nom"]."</td></
tr>";
        }
        echo "</table>";
} else {
   echo "aucun résultat";
}
$conn->close();
```

PHP et PDO : Update

```
try {
        $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
        // affectation de PDO au mode d'exception d'erreur
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);


        // Requête
        $sql = "UPDATE Personne SET nom='Doe' WHERE id=2";

        // Preparation de la transaction
         $stmt = $conn->prepare($sql);

         // execution requête
        $stmt->execute();

        // message du bon déroulement de la transaction
        echo $stmt->rowCount() . " l'enregistrement a été mis à jour";
   }
catch(PDOException $e)
   {
        echo $sql . "<br>" . $e->getMessage();
   }

$conn = null;
```

PHP orienté objet

- La programmation orientée objet a été introduite depuis PHP3 mais ce n'est qu'en PHP5 qu'elle a atteint sa maturité.
- PHP gère les objets de la même façon, qu'ils soient des références ou des gestionnaires : => chaque variable contient une référence vers l'objet plutôt qu'une copie de l'objet complet.

PHP orienté objet : Les classes Déclaration, Attributs et Méthodes

- Une première règle : une classe doit être déclarée dans un **fichier spécifique** php.
- Une classe encapsule des **variables** (appelées **attributs**) et des **fonctions** (appelées **méthodes**) qui fonctionnent avec ces variables. On les définit comme suit :

```
class SendMail {
        public $destinataire;
        public $objet;
        public $texte;
        public function envoyer() {
                mail ($this->destinataire, $this->objet, $this->texte);
        }
}
$message = new SendMail ();
$message->destinataire =  "iut-info@unice.fr";
$message->objet = "un mail";
$message->texte =  "au département informatique de l'IUT !";
$message->envoyer();
```

- L'accès à un attribut, ou une méthode se fait par la syntaxe fléchée : **->**

PHP orienté objet : Les classes Attributs et Méthodes

- Dans l'exemple précédent, les attributs et méthodes sont publiques (accessibles en dehors des méthodes de la classe). On peut les choisir :
  - **private** : utilisable à l'intérieur d'une classe
  - **protected** : utilisable uniquement dans les classes dérivées.

- Pour accéder à des attributs protégés en dehors de la classe, on utilise des **setters et getters** :

```
public function getDestinataire () {
    return $this->destinataire ;
}
public function setDestinataire ($qui ) {
    $this->destinataire = $qui ;
}
```

PHP orienté objet : Les classes Attributs et Méthodes

- Droits d'accès : Il y en a 3 à connaître :
  - public : tout le monde peut accéder à l'élément.
  - private : personne (à part la classe elle-même) n'a le droit d'accéder à l'élément.
  - protected : identique à private, mais les classes qui hériteront de celle-ci (les classes filles) auront quand même accès à l'élément.
- **Toutes les variables d'une classe doivent toujours être privées ou protégées**
- Voilà pourquoi on prend l'habitude de créer des fonctions get et set.

PHP orienté objet : Les classes
Attribut static

- Une propriété ou une méthode peuvent être déclarées avec l'attribut **static** .
- Elles sont utilisées comme des variables ou des fonctions classiques, indépendantes d'une instance quelconque (il n'y a plus de $this) :

```
Class SendMail {
    static $smtp = "smtp.unice.fr";
}
```

- Nous y accédons avec le nom de la classe :

```
echo  SendMail::$smtp;
```

- Les variables et méthodes "**static**" sont communes à l'ensemble des objets d'une même classe. On parle alors de propriétés ou de méthodes "de classe" puisqu'elles n'appartiennent pas à un objet en particulier.

PHP orienté objet : Les classes
Attribut static

- Exemple :

```
class Caniche {
    public static $caniches = 0;
    public function __construct() {
        ++self::$caniches;
    }
}
$froufrou = new Caniche();
$froufrette = new Caniche();
echo Caniche::$caniches;
```

=> L'exécution du code précédent aboutit à l'affichage de 2.

PHP orienté objet : Les classes
Référence et copie

- Depuis PHP5, le passage ou l'affection d'un objet se fait par référence (le & est présent implicitement) :

```
class obj {
        public $val;
        function __construct($x){
                $this->val = $x;
        }
        function __toString(){
                return "val = ".$this->a;
        }
}
$a = new obj(1);
$b = new obj(2);
$a=$b;
$b->val=3;
echo $a; // affiche val = 3
```

- Nous pouvons cloner explicitement l'objet :

```
$a = clone $b ;
```

PHP orienté objet : Les classes Constructeur

- L'instanciation d'une classe se fait avec le mot-clé **new** :

```
$monMail = new  SendMail();
```

- On peut ajouter à la classe un constructeur , avec la méthode **__construct** :

```
function __construct ($qui ,$objet ,$texte="???? " ) {
    $this->destinataire = $qui ;
    $this->objet = $objet ;
    $this->texte = $texte ;
}
```

- Le constructeur ainsi défini est appelé automatiquement à la création d'un objet :

```
$monMail = new  SendMail(
    "info@unice.fr" ,
    "CV" ,
    "Je souhaiterais m'inscrire à votre formation ..." );
```

PHP orienté objet : Les classes
Mots réservés

Mots réservés	Définition
class	Déclaration de classe
const	Déclaration de constante de classe
function	Déclaration d'une méthode
public/protected/private	Accès
self	la classe elle-même
parent	la classe parent
static	méthode de classe (statique)
extends	Héritage de classe
implements	Implémentation d'une interface

- Les mots clefs "self" et "parent" sont utiles pour accéder à une propriété ou méthode (statique ou non) de la classe elle-même ou de son parent.

PHP orienté objet : Les classes
Mots réservés

Mots réservés	Définition
__construct()	Constructeur de la classe
__destruct()	Destructeur de la classe
__set()	Déclenchée lors de l'accès en écriture à une propriété inexistante
__get()	Déclenchée lors de l'accès en lecture à une propriété inexistante
__call()	Déclenchée lors de l'appel d'une méthode inexistante de la classe
__callstatic()	Déclenchée lors de l'appel d'une méthode statique inexistante de la classe
__isset()	Déclenchée si on applique isset() à une propriété inexistante

PHP orienté objet : Les classes
Mots réservés

Mots réservés	Définition
__unset()	Déclenchée si on applique unset() à une propriété inexistante
__sleep()	Exécutée si la fonction serialize() est appliquée à l'objet
__wakeup()	Exécutée si la fonction unserialize() est appliquée à l'objet
__toString()	Appelée lorsque l'on essaie d'afficher directement l'objet : echo $object
__set_state()	Méthode statique lancée lorsque l'on applique la fonction var_export() à l'objet
__clone()	Appelé lorsque l'on essaie de cloner l'objet
__autoload()	Cette fonction n'est pas une méthode, elle est déclarée dans le scope global et permet d'automatiser les "include/require" de classes PHP

PHP orienté objet : Les classes
Fonctions utiles

Fonctions	Définitions
class_parents()	Retourne un tableau de la classe parent et de tous ses parents
class_implements()	Retourne un tableau de toutes les interfaces implémentées par la classe et par tous ses parents
get_class()	Retourne la classe de l'objet passé en paramètre
get_called_class()	Retourne le nom de la classe depuis laquelle une méthode statique a été appelée, tel que le Late State Binding le détermine.
class_exists()	Vérifie qu'une classe a été définie
get_class()	Retourne la classe d'un objet
get_declared_classes()	Liste des classes définies
get_class_methods()	Liste des méthodes d'une classe
get_class_vars()	Liste des propriétés d'une classe

PHP orienté objet : Les classes
Constantes utiles

Constantes	**Définitions**
__CLASS__	Donne le nom de la classe en cours
__METHOD__	Donne le nom de la méthode en cours

PHP orienté objet : Les classes
Fonctions & constantes utiles

- Exemple : Code pour afficher les méthodes des classes utilisées.

```
$classes=get_declared_classes ();
echo "<ul>" ;
foreach ($classes as $classe ) {
      echo "<li>$classe : <ul>" ;
      $liste=get_class_methods ($classe );
      foreach ($liste as $methode ) {
            echo "<li>$methode</li>" ;
      }
      echo "</ul></li>" ;
}
echo "</ul>" ;
```

PHP orienté objet : Héritage

- L'héritage est probablement le concept le plus important de la programmation orientée objet.
- Il consiste à se servir de classes "de base" pour construire des classes plus complètes.
- En suivant cette règle très simple :

Il y a héritage quand on peut dire :
"A est un B"

- Quelques exemples :
  - Une voiture est un véhicule (Voiture hérite de Vehicule)
  - Un bus est un véhicule (Bus hérite de véhicule)
  - Un moineau est un oiseau (Moineau hérite d'Oiseau)
  - Un corbeau est un oiseau (Corbeau hérite d'Oiseau)

PHP orienté objet : Héritage

- Exemple : création d'une classe Admin basée sur la classe Membre :
  - Admin possédera **les mêmes** méthodes et propriétés que Membre
  - Admin possède **en plus** des méthodes et propriété par rapport à Membre
- Ce qui donne pour la classe parent créée dans Membre.class.php :

```
class Membre {
        private $pseudo;
        private $email;
        private $actif;
        public function envoyerEMail($titre, $message) {
                mail($this->email, $titre, $message);
        }
        public function getPseudo() {
                return $this->pseudo;
        }
        public function setPseudo($nouveauPseudo) {
                if (!empty($nouveauPseudo) && strlen($nouveauPseudo) < 15)
                        $this->pseudo = $nouveauPseudo;
        }
}
```

PHP orienté objet : Héritage

- Et pour la classe fille Admin créée dans Admin.class.php :
  - Pour que PHP connaisse la classe Membre afin de permettre l'héritage, il est impératif d'inclure le fichier Membre.class.php au préalable.
  - Nous lui rajoutons des fonctionnalités comme par exemple la possibilité de choisir la couleur du pseudo.

- Soit le code suivant contenu dans Admin.class.php :

```
include_once('Membre.class.php');
class Admin extends Membre {
        private $couleur;
        public function setCouleur() {
                // ...
        }
        public function getCouleur() {
                // ...
        }
}
```

PHP orienté objet : Héritage

- Nous avons donc maintenant 2 classes : Membre et Admin.
  - Avec Membre, on peut manipuler un pseudo, un e-mail, une signature et un état actif ou non.
  - Avec Admin, on peut manipuler les mêmes choses : un pseudo, un e-mail, une signature et un état actif ou non... mais aussi de nouvelles propriétés, comme la couleur du pseudo.
- Nous pouvons maintenant créer des membres mais aussi des admins:

```
$membre = new Membre(31); // Contient un pseudo, un e-mail...
$webmaster = new Admin(2); // Données qu'un membre + la couleur
$membre->setPseudo('Arckintox'); // OK
$webmaster->setPseudo('Main'); // OK
$membre->setCouleur('Rouge'); // KO : un membre n'a pas de  couleur
$webmaster->setCouleur('Rouge'); // OK
```

PHP orienté objet : Héritage Multiple

- Il est également possible d'effectuer de multiples héritages en PHP :

```
class Animal{}
class Animal_Chien extends Animal{}
class Animal_Chien_Caniche extends
Animal_Chien{}
class Animal_Chien_Labrador extends
Animal_Chien{}
...
```

PHP orienté objet : Les droits d'accès

- Un concept très important de la POO est l'**encapsulation**. Mais avant de présenter ce principe il est important de parler des droits d'accès.
- Les droits d'accès indiquent si l'utilisateur d'une classe a le droit d'accéder directement à un élément de la classe ou non.
- Il y a 3 droits d'accès à connaître :
  - public : tout le monde peut accéder à l'élément.
  - private : personne (à part la classe elle-même) n'a le droit d'accéder à l'élément.
  - protected : identique à private, mais les classes qui hériteront de celles-ci (les classes filles) auront quand même accès à l'élément.

PHP orienté objet : Les droits d'accès

- Exemple, champs public et private :

```
class Membre {
        private $pseudo;
        private $email;
        private $signature;
        private $actif;
        public function getPseudo() {
                ...
        }
        public function setPseudo($nouveauPseudo) {
                ...
        }
}
```

La personne qui utilise la classe à travers des objets peut **uniquement accéder aux champs public** et non privés d'où :

- $membre->setPseudo('M@teo21'); // est OK, mais
- $membre->pseudo = 'M@teo21'; // est interdit

PHP orienté objet : Les droits d'accès

- Et qu'advient des champs protected?
- Ils sont identiques à ceux en private mais différents en cas d'héritage.
- En reprenant l'exemple précédent modifions la propriété du champ « email » :

  protected email;
- Dans ce cas les classes comme Admin qui héritent de Membre ont le droit d'y accéder, alors qu'en private cela serait resté impossible.

PHP orienté objet : L'encapsulation

- Lors de la définition des champs d'une classe une autre règle importante à retenir est :
  Toutes les variables d'une classe doivent toujours être privées ou protégées
- Cela évite par exemple de faire des erreurs du type :

```
$membre = new Membre(4);
$membre->email = 'Dupont';
```

  alors que 'Dupont' n'est pas un email.
- C'est la raison pour laquelle nous créons des fonctions get et set.
- Le but de l'encapsulation est de masquer à la personne qui utilise la classe son fonctionnement interne sans se soucier des variables qu'il contient.

PHP orienté objet : Résolution de portée

- Les mots clefs "**parent**" et "**self**" combinés à l'opérateur "**::**" résolvent respectivement la classe dont ils héritent et leur propre classe :

```
class Chien {
        protected function aboyer() {
                return 'Je suis un chien';
        }
}
class Chien_Labrador extends Chien {
        protected function aboyer() {
                return 'Je suis un labrador';
        }
        public function identifierParent() {
                return parent::aboyer();
        }
        public function identifierSelf() {
                return self::aboyer();
        }
}
$medor = new Chien_Labrador();
echo $medor->identifierParent().'<br/>';
echo $medor->identifierSelf().'<br/>';
```

Affiche :

Je suis un chien
Je suis un labrador

PHP orienté objet : Résolution de portée

- Les variables et méthodes "**static**" sont communes à l'ensemble des objets d'une même classe au moyen de l'opérateur "::".
- On parle alors de propriétés ou de méthodes "de classe" puisqu'elles n'appartiennent pas à un objet en particulier.

```
class Caniche {
        public static $caniches = 0;
        public function __construct() {
                ++self::$caniches;
        }
}
$froufrou = new Caniche();
$froufrette = new Caniche();
echo Caniche::$caniches;
```

Affiche : 2

PHP orienté objet : Résolution de portée

- Dans le cas où le champ $caniches était déclaré en protected soit :

  protected static $caniches = 0;

- Alors l'exécution du code :

  $froufrou = new Caniche();

  $froufrette = new Caniche();
  echo Caniche::$caniches;

- Déclenche l'erreur suivante :
  « Fatal error: Cannot access protected property Caniche::$caniches »

PHP orienté objet : Les interfaces

- Une **interface** est un ensemble de méthodes que les classes doivent définir si elles veulent l'implémenter.
- Ne pas implémenter toutes les méthodes de l'interface cause une erreur fatale.

```
interface Joueur {
        // pour implémenter l'interface Joueur on définit la méthode :
        public function jouer();
 }
 class Labrador implements Joueur{}
        // déclaration illégale car pas de définition de jouer()  dans Labrador
 $medor = new Labrador();
 $medor->jouer();
```

- Provoque l'erreur : *« Fatal error: Class Labrador contains 1 abstract method and must therefore be declared abstract or implement the remaining methods »*

PHP orienté objet : Les interfaces

- Il faut donc **surcharger** les méthodes décrites dans l'interface :

```
interface Joueur {
        // une classe qui veut implémenter
        // l'interface Joueur doit définir la méthode jouer()
        public function jouer();
 }
 class Labrador implements Joueur {
        public function jouer() {
              echo 'Ouah!';
        }
 }
 $medor = new Labrador();
 $medor->jouer();
```

PHP orienté objet : Les interfaces

- De nombreuses interfaces intéressantes sont définies dans la librairie PHP standard accessible par le lien : http://php.net/spl

- Obtenir la liste des interfaces déclarées :

```
print_r(get_declared_interfaces());
```

PHP orienté objet : Référence et clonage

- Depuis PHP 5, les objets sont tous des *références*
- Copier un objet vers un autre au moyen de l'opérateur "=" ne duplique pas l'objet, mais il créé une deuxième référence vers le même objet.
- Soit : $object = new stdClass();
- La variable **$object** contient maintenant une référence vers un objet de la classe **stdClass**, mais elle ne contient pas l'objet lui-même.

PHP orienté objet : Référence et clonage

- Donc si : $object = new stdClass();
- Alors : unset($object); implique la destruction de la seule référence donc l'objet n'existe donc plus en mémoire.
- Mais si :
  $object_1 = new stdClass();
  $object_2 = $object_1;
- Alors en faisant : unset($object_1); l'objet va persister en mémoire puisque la référence $object_2 existe encore.
- Pour que l'objet soit complétement détruit il faut en plus faire : unset($object_2);

PHP orienté objet : Référence et clonage

- Pour nous en convaincre, essayons le script suivant :

```
class Test {
        public function __destruct() {
                echo 'Objet détruit<br/>’;
        }
}
$obj_1 = new Test();
$obj_2 = $obj_1;
$obj_3 = $obj_2;
echo 'Marqueur n° 1<br/>';
unset ($obj_1);
echo 'Marqueur n° 2<br/>';
unset ($obj_2);
echo 'Marqueur n° 3<br/>';
unset ($obj_3);
```

PHP orienté objet : Référence et clonage

- L'exécution du code precedent affiche :

  Marqueur n° 1

  Marqueur n° 2

  Marqueur n° 3

  Objet détruit

- La phrase "Objet détruit" apparaît uniquement lorsque la fin du script soit à la destruction de la dernière référence.

PHP orienté objet : Référence et clonage

- Voyons maintenant avec des clones de l'objet initial :

```
class Test {
        public function __destruct() {
                echo 'Objet détruit<br/>';
        }
}
$obj_1 = new Test();
$obj_2 = clone $obj_1;
$obj_3 = clone $obj_2;
echo 'Marqueur n° 1<br/>';
unset($obj_1);
echo 'Marqueur n° 2<br/>';
unset($obj_2);
echo 'Marqueur n° 3<br/>';
```

- L'exécution du code précédent affiche :

```
Marqueur n° 1
Objet détruit
Marqueur n° 2
Objet détruit
Marqueur n° 3
Objet détruit
```

PHP orienté objet : Late Static Bindings (LSB)

- Les LSB sont une nouvelle faculté de PHP de résoudre les appels statiques plus tard dans la chronologie des évènements.
- Exemple : définissons les classes Chien et Chien_Labrador qui héritent de Chien et considérons l'emploi des méthodes « aboyer ».

```
class Chien {
        protected function aboyer() {
                return 'Je suis un chien';
        }
        public function identifier() {
                return self::aboyer(); //appel à la classe elle-même
        }
}
class Chien_Labrador extends Chien {
        protected function aboyer() {
                return 'Je suis un labrador';
        }
}
```

PHP orienté objet : Late Static Bindings (LSB)

```
$medor = new Chien();
$felix = new Chien_Labrador();
echo $medor->identifier().'<br/>';
echo $felix->identifier().'<br/>';
```

- L'exécution du code précédent affiche :
  - Pour Médor : Je suis un chien
  - Pour Félix : Je suis un chien
- On constate que la méthode **Chien::identifier()** est bien transmise par héritage à la classe **Chien_Labrador** mais cette dernière n'appelle pas sa propre fonction.

PHP orienté objet : Late Static Bindings (LSB)

- Les Late Static Bindings permettent d'utiliser un nouveau mot clef "**static::**" pour résoudre correctement la portée statique avec le mot clé **static**.
- Reprenons le code précédent :

```
class Chien {
        protected function aboyer() {
                return 'Je suis un chien';
        }
        public function identifier() {
                return static::aboyer(); //appel statique
        }
}
class Chien_Labrador extends Chien {
                protected function aboyer() {
                        return 'Je suis un labrador';
                }
}
```

- L'exécution du code précédent affiche :
  - Pour Médor : Je suis un chien
  - Pour Félix : Je suis un labrador

PHP orienté objet : Late Static Bindings (LSB)

- Dans cet exemple, "**static::**" permet de négocier l'appel de la méthode **aboyer()** au moment de l'exécution, et ainsi d'utiliser la méthode **Chien_Labrador::aboyer()**
- Une application particulièrement intéressante des LSB est dans le cadre de projets utilisant des bases de données.

```
class Table {
        static function getByPk($id) {
                return 'SELECT * FROM '.get_called_class()  WHERE id = '.(int)$id;
        }
}
class Album extends Table{}
class Artist extends Table{}
echo Album::getByPk(3);
echo Artist::getByPk(6);
```

- L'exécution du code précédent affiche :
  - Pour Album : SELECT * FROM Album WHERE id = 3
  - Pour Artist : SELECT * FROM Artist WHERE id = 6

Les Espaces de Noms

- Les espaces de noms, **namespaces**, sont un moyen de résoudre les collisions de noms de constantes, fonctions et classes.
- Prenons un exemple, nous souhaitons créer une classe nommée "Filter ». Or, ce nom est sans doute déjà utilisé par PHP ou par l'une des bibliothèques incluses dans nos scripts.
- La solution classique est de préfixer le nom de la classe de manière à le rendre unique : « MyFilter ».
- Le concept des namespaces permet de définir des noms de classes, fonctions *etc*. au sein d'un espace de noms et évite l'utilisation de préfixe en définissant par exemple le namespace « MyFilterSpace ».
- Notre namespace peut alors contenir la classe « Filter » même si elle est au préalable définie dans PHP sans qu'il y ait d'erreur de compilation.

Les Espaces de Noms

- Déclaration :

  namespace <Nom>;

  ...
- Utilisation, exemple de la définition d'un alias:

  namespace Models;

  use Models **as** M;
- Dans ce namespace les deux appellations sont équivalentes :
  - $object = new Models::MyObj(); //avec le nom complet
  - $object = new M::MyObj(); //avec l'alias

Les Espaces de Noms

- Les espaces de noms composés peuvent être importés tels quels ou avec un alias :

```
namespace Cours::Models;
...
use Cours::Models;
use Cours::Models as M;
...
```

Les Espaces de Noms

- Cependant, les noms qui ne sont pas composés doivent être importés avec un alias :

```
namespace Models;
...
use Models as M; ...
```

- Erreur si on ne définit pas d'alias :

```
use Models;
...
```

Warning: The use statement with non-compound name 'Models' has no effect in C:\Web\online\http\cours-php\namespaces\index.php on line 6

Les Espaces de Noms

- Il est possible d'imbriquer les espaces de noms :

```
namespace Offline::Sites::Application::Models;
use Offline::Sites::Application::Models as M;
$object = new M::MyObj();
```

- Si nous sommes dans le namespace Cours nous pouvons nous abstenir d'utiliser le préfixe "Cours::" devant les symboles de cet espace de noms.
- Car PHP suppose que vous utilisez des symboles du même espace de noms, ou bien un symbole du langage:
  - Si nous utilisons un symbole du scope global (mais pas interne à PHP), le nom du symbole est préfixé par "::".
  - Si nous utilisons un symbole d'un autre espace de noms, il faut bien entendu préfixer votre symbole de l'espace de nom complet.

Les Espaces de Noms : Exemples

- Exemples d'utilisation : Reprenons un contexte MVC assez simple pour illustrer une utilité possible des namespaces.

- Ajoutons à "httpd.conf" :

```
<Directory "C:/Web/online/http/cours-php/namespaces">
    AllowOverride None
    php_value include_path ".;C:/Web/offline/sites/cours-php/namespaces"
    SetEnv HTTP_ROOT /cours-php/namespaces/
    RewriteEngine on
    RewriteCond %{REQUEST_URI} !\.(js|css|jpg|png|gif)$
    RewriteRule .* index.php
</Directory>
```

Les Espaces de Noms : Exemples

- controllers/member.php :

```
namespace Cours::Controllers;
class Member {
        public static function whoAmI() {
                return 'Je suis un Contrôleur';
        }}
```

- models/member.php :

```
namespace Cours::Models;
class Member {
        public static function whoAmI() {
                return 'Je suis un Modèle';
                }}
```

- views/member.php :

```
namespace Cours::Views;
class Member {
        public static function whoAmI() {
                return 'Je suis une Vue';
                }}
```

Les Espaces de Noms : Exemples

- index1.php :

```
require 'models/member.php';
require 'views/member.php';
require 'controllers/member.php';
header('Content-Type: text/html; charset=utf-8');
echo Cours::Models::Member::whoAmI().'<br/>';
echo Cours::Views::Member::whoAmI().'<br/>';
echo Cours::Controllers::Member::whoAmI();
```

- index2.php :

```
require 'models/member.php';
require 'views/member.php';
require 'controllers/member.php';
use Cours::Models;
use Cours::Views;
use Cours::Controllers;
header('Content-Type: text/html; charset=utf-8');
echo Models::Member::whoAmI().'<br/>';
echo Views::Member::whoAmI().'<br/>';
echo Controllers::Member::whoAmI();
```

Les Espaces de Noms : Exemples

- index3.php :

```
require 'models/member.php';
require 'views/member.php';
require 'controllers/member.php';
use Cours::Models as M;
use Cours::Views as V;
use Cours::Controllers as C;
header('Content-Type: text/html; charset=utf-8');
echo M::Member::whoAmI().'<br/>';
echo V::Member::whoAmI().'<br/>';
echo C::Member::whoAmI();
```

- L'exécution de ces codes affichent tous :
  - Je suis un Modèle
  - Je suis une Vue
  - Je suis un Contrôleur

Gestion des Erreurs

Dans ce chapitre nous traitons les méthodes de vérification d'erreur les plus courantes en PHP comme :

- L'instruction faisant appel à la fonction die(),
- Les erreurs particulières ou les erreurs déclenchées,
- Le rapport d'erreurs.

Traitement basique des erreurs : la fonction die ou exit :

```
if (!file_exists("welcome.txt"))
    die("Fichier introuvable ou inexistant");
else ...
```

=> Fin du programme lorsque l'erreur est constatée.

Gestion des Erreurs : Création d'un Gestionnaire

- Consiste à créer une fonction spécifique à laquelle nous ferons appel lorsqu'une erreur survient.
- Cette fonction comporte au moins deux variables dont l'une est le niveau d'erreur et l'autre est le message d'erreur :

  fonction_erreur(niveau_erreur,message_erreur,

  [fichier_erreur,ligne_erreur,contexte_erreur])

Gestion des Erreurs : Création d'un Gestionnaire

- Description des paramètres d'appel

Paramètre	Description
niveau_erreur	Indique le niveau de l'erreur définie par le programeur. Ce paramètre doit être un nombre, la table ci-dessus indique les valeurs possibles pour le niveau de l'erreur
message_erreur	Indique le message pour l'erreur définie par le programmeur.
fichier_erreur	Optionnel. Indique le nom du fichier dans lequel l'erreur a eu lieue
ligne_erreur	Optionnel. Indique le numéro de la ligne où l'erreur a eu lieue
contexte_erreur	Optionnel. Indique une table contenant chaque variable et leur(s) valeur(s) utilisées lorsque l'erreur est survenue

Gestion des Erreurs : Création d'un Gestionnaire

- Les niveaux d'erreurs

Val.	Constante	Description
2	E_WARNING	Erreur d'exécution non-fatale. L'exécution du script n'est pas stoppée.
8	E_NOTICE	Notice d'exécution. Le script a trouvé quelque chose qui semble être une erreur, mais qui pourrait aussi arriver lorsque le script s'exécute normalement
256	E_USER_ERROR	Erreur fatale définie par le programmeur: une E_ERROR est affectée par le programmeur dans la fonction PHP trigger_error()
512	E_USER_WARNING	Erreur non-fatale définie par le programmeur: un E_WARNING est affecté par le programmeur dans la fonction PHP trigger_error()
1024	E_USER_NOTICE	Note d'exécution définie par le programmeur: un E_NOTICE est affecté par le programmeur dans la fonction PHP trigger_error()
4096	E_RECOVERABLE_ERROR	Erreur fatale que l'on peut encapsuler. Elle s'apparente à une E_ERROR qui peut être encapsulée dans un gestionnaire d'erreur défini par le programmeur (voir aussi la fonction set_error_handler())
8191	E_ALL	Toutes les erreurs et les warnings, à l'exception du niveau E_STRICT

Gestion des Erreurs : Création d'un Gestionnaire

- Exemple d'une fonction de gestion d'erreur :

```
function customError($errno, $errstr) {
        echo "<b>Erreur:</b> [$errno] $errstr<br />";
        echo "Fin Script"; die();
}
```

- Lorsqu'elle est déclenchée, cette fonction doit recevoir le niveau et le message d'erreur qu'elle affiche et ensuite termine le script.

Gestion des Erreurs : Création d'un Gestionnaire

Affectation du gestionnaire d'erreur

- Pour la durée du script, l'affectation de la fonction précédente comme le gestionnaire d'erreur par défaut se fait à l'aide de la fonction **set_error_handler** :

```
set_error_handler("customError");
```

- La fonction set_error_handler n'a qu'un seul paramètre en entrée dans le cas d'autres erreurs, il suffit de passer les autres fonctions d'erreur en paramètre.

Gestion des Erreurs : Déclenchement des Erreurs

- Les erreurs sont déclenchées grâce à la fonction **trigger_error** :

```
$test=2;
if ($test>1) {
    trigger_error("La valeur doit être inférieure ou égale à 1");
}
```

- Les erreurs sont déclenchées si $test >1. L'exécution du code affiche :

Notice: La valeur doit être inférieure ou égale à 1
in **C:\web\test.php** on line **6**

Gestion des Erreurs : Déclenchement des Erreurs

- Une erreur peut être déclenchée dans un endroit quelconque d'un script par l'ajout d'un second paramètre à trigger_error() pour spécifier le niveau d'erreur déclenché.
- Exemple complet :

```
function customError($errno, $errstr) {
        echo "<b>Erreur:</b> [$errno] $errstr<br />";
        echo "Fin du Script";
        die(); }
//affectation du gestionnaire d'erreur
set_error_handler("customError",E_USER_WARNING);
//déclenchement d'une erreur
$test=2;
if ($test>1)
        trigger_error("La valeur doit être <= 1",E_USER_WARNING);
```

=> **Erreur:** [512] La valeur doit être <= 1
Fin du Script

Gestion des Erreurs : Stockage des Erreurs

- Par défaut PHP envoie les erreurs au système de stockage d'erreurs du serveur : error.log
- La fonction **error_log**() permet de stocker l'erreur dans un fichier spécifique ou de l'envoyer à une destination spécifique.
- Exemple : envoi des erreurs par mail.

```
customError($errno, $errstr) {
    echo "<b>Erreur:</b> [$errno] $errstr<br />";
    echo "Le webmaster a été averti";
    error_log("Erreur: [$errno] $errstr",1,
        "tony@unice.fr","From: webmaster@unice.fr");
}
```

Gestion des Erreurs : Stockage des Erreurs

- Exemple suite ...

```
// affectation du gestionnaire d'erreur
set_error_handler("customError",E_USER_WARNING);
//déclenchement d'erreur
$test=2;
if ($test>1)
   trigger_error("La valeur doit être <= 1",E_USER_WARNING);
```

- L'exécution du code précédent affiche:

  Erreur: [512] La valeur doit être <= 1

  Le webmaster a été averti

- Quand au mail reçu par le déclenchement de l'erreur, il contient le message suivant:

  Erreur: [512] La valeur doit être <= 1

Gestion des Erreurs : Gestion des Exceptions

- Le déclenchement d'une exception implique les actions suivantes :
  - La sauvegarde du contexte du code,
  - L'exécution d'une fonction spécifique pour le traitement de l'exception,
  - D'après la situation cette fonction spécifique peut reprendre l'exécution en partant du contexte sauvegardé du code, terminer l'exécution du script ou continuer le script a partir d'un autre emplacement dans le code.

Gestion des Erreurs : Gestion des Exceptions

- Nous allons maintenant montrer différentes méthodes de traitement des exceptions par :
  - L'utilisation basique des exceptions,
  - La création d'une fonction spécifique pour le traitement de l'exécution,
  - La technique d'exceptions multiples,
  - La relance d'exception,
  - La mise en place d'un gestionnaire d'exception de haut niveau.

Gestion des Erreurs : Gestion des Exceptions

L'utilisation basique des exceptions

- Lorsqu'une exception est lancée, le code courant ne sera plus exécuté ; PHP tente alors de capturer l'exception et d'afficher les informations qui lui sont relatives.
- Dans ce cas un message d'erreur indiquant « Exception Introuvable » est émis.

Gestion des Erreurs : Gestion des Exceptions

- Exemple, lancement d'une exception qui ne peut être capturée.

```
function checkNum($number) {
        if($number>1) {
                throw new Exception("La valeur doit être <= 1");
        }
        return true;
}
//déclencher l'exception
checkNum(2);
```

- L'exécution du code précédent génère l'erreur suivante:

Fatal error: Uncaught exception 'Exception' with message 'La valeur doit être <= 1' in C:\web\test.php:6

Stack trace: #0 C:\web\test.php(12):

checkNum(28) #1 {main} thrown in **C:\webfolder\test.php** on line **6**

Gestion des Erreurs : Gestion des Exceptions

Le bloc try, l'instruction throw et le bloc catch

- Pour éviter l'erreur survenue dans l'exemple précédent, la programmation d'une exception doit contenir un bloc « try », une instruction « throw » et un troisième bloc « catch » :
  - Le bloc « try » permet d'appeler la fonction contenant l'exception. Si l'exception n'est pas déclenchée le code se poursuit normalement, dans le cas contraire une exception est lancée.
  - L'instruction « throw » permet de lancer l'exception. A chaque instruction « throw » doit correspondre au moins un bloc « catch ».
  - Le bloc catch capture l'exception. A l'intérieur du bloc catch est créé un objet Exception contenant les informations relatives à l'exception.

Gestion des Erreurs : Gestion des Exceptions

- Exemple : programmation valide d'une exception.

```
//créer une fonction avec une exception
function checkNum($number) {
        if($number>1)
                throw new Exception("La valeur doit être <= 1");
        return true;
}
//déclencher l'exception dans un bloc « catch »
try {
        checkNum(2);
        //Si l'exception est lancée, ce texte ne s'affichera pas
        echo 'Si vous voyez ce texte, le nombre est <= 1';
}
//capture de l'exception
catch(Exception $e) {
        echo 'Message: ' .$e->getMessage();
}
```

Gestion des Erreurs : Gestion des Exceptions

- L'exécution du code précédent génère le message suivant:

  Message: La valeur doit être <= 1

- Dans l'exemple précédent une exception est lancée et capturée :
  - La fonction checkNum() est créée ; elle vérifie si le nombre passé en argument est inférieur ou égal à 1. Si le nombre est supérieur à 1, une exception est lancée.
  - La fonction checkNum() est appelée dans un bloc « try ».
  - L'exception programmée à l'intérieur de la fonction est lancée, puisque le nombre passé en argument de checkNum est 2.
  - Le bloc catch capture alors l'exception et crée l'objet $e représentant l'exception.
  - Le message d'erreur contenu dans l'exception que l'on retrouve par l'exécution de la méthode $e->getMessage() est alors affiché.

Gestion des Erreurs : Gestion des Exceptions

- Cependant, il est possible qu'une exception déclenchée ne puisse pas être capturée par un catch, c'est à dire qu'un bloc catch n'ait pas été prévu pour cette exception.
- Afin de respecter la règle « **chaque exception lancée doit pouvoir être capturée dans un catch** », il est préférable de mettre en place **un gestionnaire d'exception** de haut niveau, qui traiterait les erreurs résultantes des exceptions qui n'ont pu être capturées.

Gestion des Erreurs : Gestion des Exceptions

Création d'une classe Exception personnalisée

- Il suffit de créer une classe spéciale comportant des fonctions qui peuvent être appelées lorsqu'une exception survient.
- Cette classe hérite des propriétés de la classe Exception à laquelle s'ajoute des fonctions spécifiques à l'exception à traiter.

Gestion des Erreurs : Gestion des Exceptions

- Exemple

```
class customException extends Exception {
        public function errorMessage() {
                //message d'erreur
                $errorMsg = 'Erreur à la ligne '.$this->getLine().' dans '.$this->getFile().': <b>'.
                                $this->getMessage().'</b> n'est pas une adresse E-Mail valide';
                return $errorMsg;
        }
}
$email = "someone@example...com";
try {
        //verifier si l'E-mail est valide
        if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
                //lancer une exception si si l'E-mail est invalide
                throw new customException($email);
        }
}
catch (customException $e) {
        //afficher le message personnalisé
        echo $e->errorMessage();
}
```

Gestion des Erreurs : Gestion des Exceptions

- Dans l'exemple précédent nous pouvons observer :
  - La classe customException() est créée comme une extension de l'ancienne classe Exception. Ainsi, elle hérite des méthodes et des propriétés de cette dernière.
  - Une fonction errorMessage() est créée. Cette fonction retourne un message d'erreur dans le cas où l'adresse e-mail est invalide.
  - La variable $email est affectée à une adresse e-mail non valide.
  - Le bloc « try » est exécuté, une exception est lancée puisque l'adresse e-mail est invalide.
  - Le bloc « catch » capture l'exception et affiche le message d'erreur.

Gestion des Erreurs : Gestion des Exceptions Multiples

Les exceptions multiples

- Il est possible de lancer plusieurs exceptions correspondantes à plusieurs conditions.
- Pour traiter ces exceptions nous pouvons utiliser plusieurs blocs if … else, un switch, ou avoir recours aux exceptions multiples.

Gestion des Erreurs : Gestion des Exceptions Multiples

- Reprenons l'exemple précédent :

```
$email = someone@exemple.com;
try {
        try {
                //vérification de la chaine "exemple" dans l'email
                if(strpos($email, "exemple") !== FALSE) {
                        //lancer une exception si l'e-mail n'est pas valide
                        throw new Exception($email);
                }
        }
        catch(Exception $e) {
                //relancer l'exception
                throw new customException($email);
        }
}
catch (customException $e) {
        //afficher le message correspondant
        echo $e->errorMessage();
}
```

Gestion des Erreurs : Gestion des Exceptions Multiples

- Par rapport à la gestion d'une exception simple nous remarquons que :
  - Le bloc « try » contient un autre bloc « try » pour pouvoir relancer l'exception.
  - L'exception est déclenchée puisque l'e-mail contient la chaine « exemple ».
  - Le bloc « catch » capture l'exception et relance une « customException ».
  - L'exception « customException » est capturée à son tour et affiche le message d'erreur.
- Dans le cas où l'exception n'est pas capturée dans son bloc « try » courant, elle cherche à être capturée par un bloc d'exception de plus haut niveau.

Gestion des Erreurs : Gestion des Exceptions Multiples

Mise en place d'un gestionnaire d'exception « top level » (de haut niveau)

- La fonction **set_exception_handler**() permet la mise en place d'une fonction définie par le programmeur pour gérer les exceptions qui n'ont pas été capturées.
- Exemple :

```
function myException($exception) {
        echo "<b>Exception:</b> " , $exception->getMessage();
}
set_exception_handler('myException');
throw new Exception('Une exception non-capturée est survenue');
```

- L'exécution de ce code affiche le message suivant :

  Exception : Une exception non-capturée est survenue

Gestion des Erreurs : Gestion des Exceptions Multiples

- Dans le code ci-dessus il n'y a pas de bloc de capture. Au lieu de cela, le gestionnaire d'exception « top level » est déclenché. Cette fonction pourra être utilisée pour capturer les exceptions qui n'ont pu l'être.

Les règles pour les exceptions :

- Le code doit être encapsulé dans un bloc «try», pour permettre de capturer toute exception potentielle.
- Tous les blocs «try» où une ou plusieurs exceptions sont censées être lancées, doivent avoir au moins un bloc de capture correspondant.
- Des blocs de captures multiples doivent être utilisés pour capturer différentes classes d'exception.
- Les exceptions peuvent être lancées (ou relancées) dans un bloc de capture au sein d'un bloc «try».