Hypertext Preprocessor

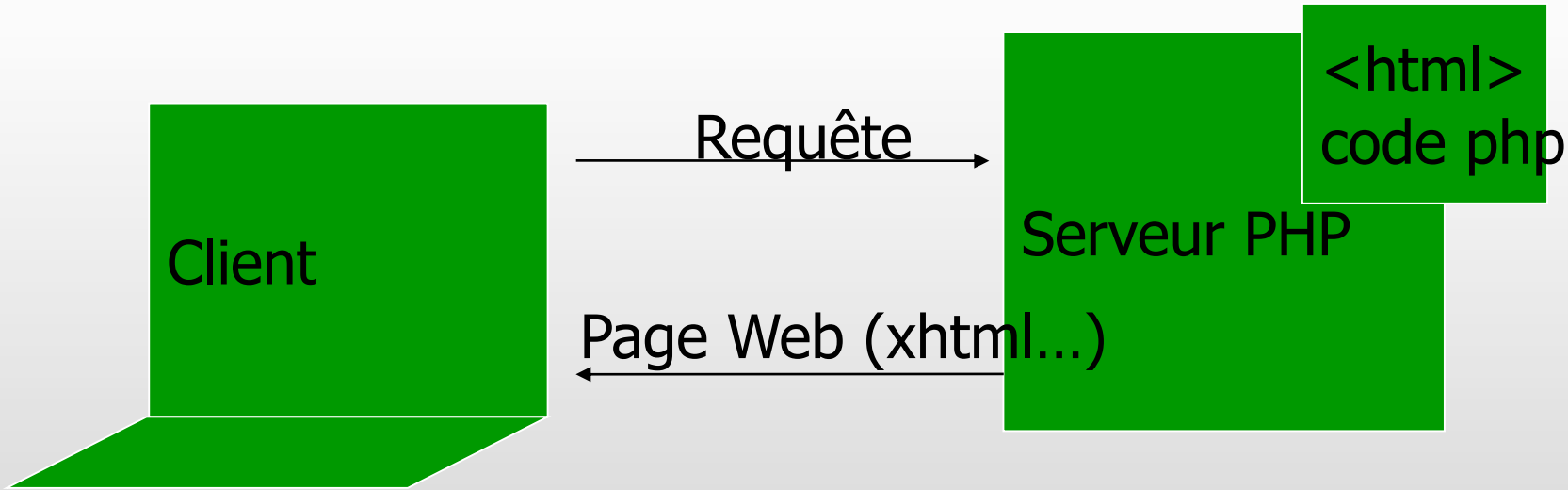
Université Sultan Moulay Slimane
Faculté poly disciplinaire Khouribga
SMI/S6

Remerciements

- Le matériel des diapositives est pris de différentes sources comprenant :
  - https://www.php.net/manual/fr/language.variables.scope.php
  - https://waytolearnx.com/
  - https://www.w3schools.com/php/func_array_count_values.asp
  - https://www.chiny.me/objet-pdo-pour-la-connexion-a-une-base-de-donnees-en-php-8-12.php

PHP

- PHP: Hypertext Preprocessor
- Code exécuté coté serveur
- Langage interprété [compilable]

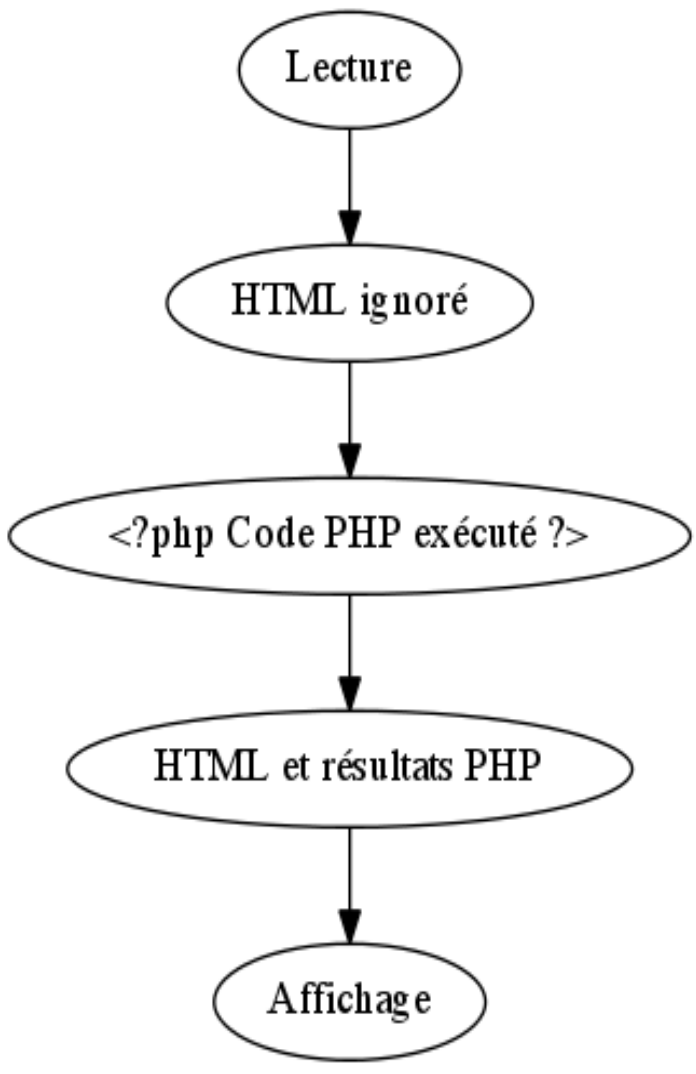


PHP

- Portabilité et distribution C/S, 3/3, P2P
- Accès aux bases de données facile
- Génération graphique sophistiquée
- Interopérabilité avec XML, parsing xml avec les standards SAX et DOM, Web services…
- Support des protocoles réseau (LDAP, IMAP, SNMP, POP3…)
- Traitement des chaînes de caractères (regex, perl)
- Nombreuses bibliothèques (PEAR, PECL, ...)
- Syntaxe proche du C
- Interpréteur souple
- Types de données classiques
- Typage à première affectation
- Conversion automatique de type
- Tableaux dynamiques à sélecteur
- Modèle objet intégré

Fonctionnement

- **L'interpréteur**
  - lit un fichier source .php puis génère un flux de sortie avec les règles suivantes :
    - toute ligne située à l'extérieur d'un bloc PHP (entre <?php et ?>) est recopiée inchangée dans le flux de sortie
    - le code PHP est interprété et génère éventuellement des résultats intégrés eux aussi au flux de sortie
    - les erreurs éventuelles donnent lieu à des messages d'erreurs qu'on retrouve également dans le flux de sortie (selon la configuration du serveur)
    - une page html pure sauvegardée avec l'extension .php sera donc non modifiée et renvoyée telle quelle . .



Historique

L'origine de PHP remonte à 1995 quand Rasmus Lerdorf a créé PHP/FI, une librairie de scripts Perl.

Au fur et à mesure des évolutions, la librairie a été portée en C et agrémentée de nouvelles fonctionnalités pour créer des pages dynamiques simples pour le web et accéder à quelques sources de données.

PHP/FI signifie Personal Home Page/Forms Interpreter. PHP/FI 2.0 sort en 1997.

PHP 3.0 sort en juin 1998, c'est la première version développée conjointement par Rasmus Lerdorf, Andi Gutmans et Zeev Suraski et entièrement réécrite.

PHP 4 sort officiellement en mai 2000 et apporte des performances accrues, le supports de plusieurs serveurs web, les sessions, une sécurité accrue.

Et PHP 5 ?, PHP 5 est une amélioration des performances du moteur Zend (Zend Engine 2), un modèle objet étendu et très proche du C++, une compatibilité accrue avec de nouveaux standards et les anciens (RPC, XML, .NET, ...).

Aujourd’hui on ai en version 7 du PHP

Editeurs

- The best Free Code Editors for Windows & Mac:
  - Notepad++
  - Atom
  - Visual Studio Code (sublime text is a payed version)
  - Brackets
    - Extention: file --->extension manager
      - Emmet (extending CSS, HTML Code)
      - Autoprefixer
      - Autosaver
      - Bracket icons

Variables

- Une variable commence par un dollar « $ » suivi d'un nom de variable. Les variables ne sont pas typées au moment de leur création. Attention PHP est sensible à la casse : var et Var ne sont pas les mêmes variables ! Voici les règles à respecter :
  - Une variable peut commencer par une lettre
  - Une variable peut commencer par un souligné (underscore) « _ »
  - **Une variable ne doit pas commencer par un chiffre**.
- // Déclaration et règles
- $var=1; // $var est à 1
- $Var=2; // $ Var est à 2
- $_toto='Salut'; // Ok
- $3petitscochons=5; // Invalide : commence par un chiffre

Portée d'une variable

- La portée d'une variable dépend du contexte. Une variable déclarée dans un script et hors d'une fonction est globale mais par défaut sa portée est limitée au script courant, ainsi qu'au code éventuellement inclus (include, require) et n'est pas accessible dans les fonctions ou d'autres scripts

```
$a=1; // globale par défaut
function foo() {
echo $a; // c'est une variable locale à la fonction : n'affiche rien
}
```

Pour accéder à une variable globale dans une fonction, il faut utiliser le mot-clé global.

```
$a=1; // globale par défaut
$b=2; // idem
function foo() {
    global $a,$b; // on récupère les variables globales
    $b=$a+$b; }
 echo $b; // affiche 3
```

Variables(2)

- PHP accepte les variables statiques. Comme en C une variable statique ne perd pas sa valeur quand on sort d'une fonction.

```
function test_static() {
        static $a=0;
        echo $a; // +1 à chaque passage dans la fonction
         $a++; }
```

Variables prédéfinies

PHP dispose d'un grand nombre de variables prédéfinies. Ces variables sont généralement de type scalaires ou des tableaux. Elles sont souvent de type superglobales, c'est à dire accessible depuis n'importe où sans notion de portée. Voici quelques tableaux prédéfinis (voir au point Tableaux pour comprendre leur utilisation).

- $_GLOBALS : tableau des variables globales. La clé est le nom de la variable.
- $_SERVER : variables fournies par le serveur web, par exemple 'SERVER_NAME'
- $_GET : variables fournies par HTTP par la méthode GET (formulaires)
- $_POST : idem mais pour la méthode POST
- $_COOKIE : les variables fournies par un cookie
- $_FILES : variables sur le téléchargement d'un fichier (upload)
- $_ENV : accès aux variables d'environnement du serveur
- $_SESSION : les variables de session (voir cours sur les sessions)

Variables dynamiques

- Une variable dynamique utilise la valeur d'une variable comme nom d'une autre variable. On utilise les variables dynamiques en rajoutant un « $ » devant le nom de la première variable.

```
$a="var";
$$a=1; // $$a=1 equivaut en fait à $var=1
echo $a; // affiche var
echo $$a; // affiche 1
echo $var; // affiche 1
```

Attention avec les tableaux ! Pour éviter toute ambiguïté, il est préférable de placer la variable entre accolades.

Types de variables

- Quatre types de données simples :
  - Integer
    - Entier compris entre -2 147 483 647 et 2 147 483 647.
  - double
    - nombre à virgule flottante compris entre -1.78*10308 et +1.78*10308.
  - String
    - chaîne de caractères de longueur variable 'string' ou "$string".
  - Boolean
    - valeur booléenne true ou false.
- Les variables ne possèdent pas de types de données prédéfinis et changent de type selon leur contenu
  - $a = 1; $a = 1.3 ; $a = ‘ABC’ ;

- Voici le tableau issu de la documentation PHP des séquences pouvant être utilisés avec les guillemets doubles.

Séquence	Valeur
\n	Nouvelle ligne (linefeed, LF ou 0x0A (10) en ASCII)
\r	Retour à la ligne (carriage return, CR ou 0x0D (13) en ASCII)
\t	Tabulation horizontale (HT ou 0x09 (9) en ASCII)
\\	Antislash
\$	Caractère $
\"	Guillemets doubles
\[0-7]{1,3}	Une séquence de caractères qui permet de rechercher un nombre en notation octale.
\x[0-9A-Faf]{1,2}	Une séquence de caractères qui permet de rechercher un nombre en notation hexadécimale.

PHP

- Entre guillemets simples ou doubles
  - 'Ma chaine $var' : pas interprétée
  - "Ma chaine $var " : interprétée
- Caractères de contrôle d'espace
- Caractères d'échappement \ permettant l'exécution du contrôle
  - \t tabulation
  - \n nouvelle ligne
  - \r retour à la ligne
  - \v tabulation verticale
- Possibilité d'imposer le suivi d'un format

PHP

- Tableaux classiques
  - // initialisation

```
$table=array(0,1,2,3,4,5);
```

  - // parcours simple

```
print 'table a ' . count($table) . " éléments\n";
for($i=0;$i<count($table);$i++)
    print "table[$i]=$table[$i]\n";
```

Echo vs Print

	echo	print
Paramètres	echo peut prendre plus d'un paramètre lorsqu'il est utilisé sans parenthèses. La syntaxe est la suivante: echo $arg1, $arg2 . Notez que echo($arg1, $arg2) n'est pas correct.	print ne prend qu'un seul paramètre.
Rapidité	Il est plus rapide que print car il ne renvoie aucune valeur.	Il est plus lent que echo car il renvoie une valeur.
Valeur de retour	echo ne renvoie aucune valeur	print renvoie toujours 1
Syntaxe	void echo (string $arg1 [...])	int print (string $arg)

PHP

- Les tableaux associatifs
  - Aussi appelés dictionnaires
  - Tableaux associant une clé à une valeur
  - La clé peut être une chaîne de caractère ou un nombre
  - // déclaration d'un tableau
    $dico=array('Pierre'=>20, 'Paul'=>22, 'Marie'=>18);
  - // Insertion d'une ligne
    $dico["cle"] = valeur;
    exemple : $dico['Pierre'] = 20 ;
  - // Accès à un élément
    $variable = $dico["cle"]; exemple: $age = $dico['Pierre'] ;

Tableaux

- Principe
  - Création à l'aide de la fonction array()
  - Uniquement des tableaux à une dimension
    - Les éléments d'un tableau peuvent pointer vers d'autres tableaux
  - Les éléments d'un tableau peuvent appartenir à des types distincts
  - L'index d'un tableau en PHP commence de 0
  - Pas de limites supérieures pour les tableaux
  - La fonction count() pour avoir le nombre d'éléments d'un tableau

Tableaux

- Les tableaux indicés et les tableaux associatifs
  - Tableau indicé
    - Accéder aux éléments par l'intermédiaire de numéros

```
$tableau[indice] = valeur;
$jour[3] = "Mercredi";
$note[0] = 20;
$tableau = array(valeur0, valeur1,..., valeurN);
$jour = array("Dimanche", "Lundi", "Mardi",
  "Mercredi", "Jeudi", "Vendredi", "Samedi");
$note = array(20, 15, 12.6, 17, 10, 20, 11, 18,
  19);
$variable = $tableau[indice];
$JJ = $jour[6]; // affecte "Samedi" à $JJ
echo $note[1] + $note[5];
```

Tableaux

- Les tableaux indicés et les tableaux associatifs
  - Tableau associatif (ou table de hachage)
    - Les éléments sont référencés par des chaînes de caractères associatives en guise de nom: la clé d'index

```
$tableau["indice"] = valeur;
$jour["Dimanche"] = 7
$jour["Mercredi"] = "Le jour des enfants"

$tableau = array(ind0 => val0, ind1 => val1,..., indN =>
  valN);
$jour = array("Dimanche" => 1, "Lundi" => 2, "Mardi" =>
  3, "Mercredi" => 4, "Jeudi" => 5, "Vendredi" => 6,
  "Samedi" => 7);

$variable = $tableau["indice"];
$JJ = $jour["Vendredi"]; //affecte 6 à $JJ
echo $jour["Lundi"]; //retourne la valeur 2
```

Tableaux

- Tableaux multidimensionnels
  - Pas d'outils pour créer directement des tableaux multidimensionnels
  - L'imbrication des tableaux est possible

```
$tab1 = array(Val0, Val1,..., ValN);
$tab2 = array(Val0, Val1,..., ValN);
// Création d'un tableau à deux dimensions
 $tableau = array($tab1, $tab2);
$mois  =  array("Janvier",  "Février",  "Mars",  "Avril",  "Mai",
   "Juin", "Juillet", "Août", "Septembre", "Octobre", "Novembre",
   "Décembre");
$jour = array("Dimanche", "Lundi", "Mardi", "Mercredi", "Jeudi",
   "Vendredi", "Samedi");
&element_date = array(&mois, &jour);

$variable = $tableau[indice][indice];
$MM = $element_date[0][0]; //affecte "Janvier" à $MM
echo $element_date[1][5] . " 7 " . $element_date[0][2] . "2002";
   // retourne "Jeudi 7 Mars 2002"
```

Lecture des éléments d'un tableau

- Avec une boucle for

```
for ($i=0; $i<count($tab) ; $i++){
if ($tab[$i]== "a") {echo $tab[$i], "<br  />"; }}
```

- Avec une boucle while

```
$i=0;
while ($tab[$i]){
if ($tab[$i][0] =="a"; $i++; ) {echo $tab[$i], "<br />
"; }}
```

- Avec La boucle foreach

```
$jour = array("Dimanche", "Lundi", "Mardi", "Mercredi", "Jeudi",
  "Vendredi", "Samedi");
$i = 0;
foreach($jour as $JJ) { echo "La cellule n° ". $i . " : " . $JJ .
  "<br>"; $i++; }
```

Tableaux

- Lecture des éléments d'un tableau
  - Parcours d'un tableau associatif
    - Réalisable en ajoutant avant la variable $valeur, la clé associée

Syntaxe:

```
$tableau = array(clé1 => val1, clé2 => val2, ..., cléN =>
  valN);
foreach($tableau as $clé => $valeur) {
echo "Valeur ($clé): $valeur"; }
```

Exemple:

```
$jour = array("Dimanche" => 7, "Lundi" => 1, "Mardi" => 2,
  "Mercredi" => 3, "Jeudi" => 4, "Vendredi" => 5, "Samedi" =>
  6);
foreach($jour as $sJJ => $nJJ) {
echo "Le jour de la semaine n° ". $nJJ . " : " . $sJJ .
  "<br>"; }
```

- On peut aussi créer des tableaux multidimensionnels à l'aide des deux méthodes précédentes.

```
$tab=array("un"=>array("riri",1=>"fifi",2=>'loulou'),2=>array(1,2,3),array('un', 'deux','trois'));
echo $tab['un'][0]; // riri
echo $tab[2][1]; // 2
echo $tab[3][2]; // trois
$tab2['un']['deux']='test'; // créé un tableau à deux dimensions
```

Tableaux

- Fonctions de tri
  - Tri selon les valeurs
    - La fonction sort() effectue un tri sur les valeurs des éléments d'un tableau selon un critère alphanumérique : selon les codes ASCII :
      - Le tableau initial est modifié et non récupérables dans son ordre original
      - Pour les tableaux associatifs les clés seront perdues et remplacées par un indice créé après le tri et commençant à 0
    - La fonction rsort() effectue la même action mais en ordre inverse des codes ASCII.
    - La fonction asort() trie également les valeurs selon le critère des codes ASCII, mais en préservant les clés pour les tableaux associatifs
    - La fonction arsort() la même action mais en ordre inverse des codes ASCII
    - la fonction natcasesort() effectue un tri dans l'ordre alphabétique non ASCII (« a » est avant « z » et « 10 » est après « 9 »)

Tableaux

- Fonctions de tri
  - Tri sur les clés
  - La fonction ksort() trie les ***clés*** du tableau selon le critère des codes ASCII (ascendant), et préserve les associations clé /valeur
    - La fonction krsort() effectue la même action mais en ordre inverse des codes ASCII
- Exemple

```
<?php

  $tab2=array("1622"=>"Molière","1802"=>"Hugo","1920"=>"Vian") ;
  ksort ($tab2);
  echo "<h3 >  Tri sur les clés de \$tab2 </h3>" ;
  foreach ($tab2 as $cle=>$valeur) {
         echo "<b> l'élément a pour clé : $clé; et pour valeur :
        $valeur </b> <br />";
        }
?>
```

Tableaux

- Les fonctions de tableaux

$tableau = array_count_values($variable);

retourne un tableau comptant le nombre d'occurrences des valeurs d'un tableau.

$tableau = array_diff($var_1, $var_2, ..., $var_N);

retourne dans un tableau contenant les valeurs différentes entre deux ou plusieurs tableaux.

$tableau = array_intersect($var_1, $var_2, ..., $var_N);

retourne un tableau contenant les enregistrements communs aux tableaux entrés en argument.

$tableau = array_flip($variable);

intervertit les paires clé/valeur dans un tableau.

$tableau = array_keys($variable [, valeur]);

retourne toutes les clés d'un tableau ou les emplacements d'une valeur dans un tableau.

Exemples

```
<!DOCTYPE html>
<html>
<body>
<?php
$a=array("A","Cat","Dog","A","Dog");
print_r(array_count_values($a));
?>
</body>
</html>
```

Array ([A] => 2 [Cat] => 1 [Dog] => 2)

```
<?php
$a1=array("a"=>"red","b"=>"green","c"=>"blue
","d"=>"yellow");
$a2=array("e"=>"red","f"=>"green","g"=>"blue"
);

$result=array_diff($a1,$a2);
print_r($result);
?>
```

Array ([d] => yellow)

Exemples

```
<!DOCTYPE html><html><body>
<?php
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$result=array_flip($a1);
print_r($result);
?>
</body></html>
```

→ Array ([red] => a [green] => b [blue] => c [yellow] => d)

Pour plus d'exemples:

https://www.w3schools.com/php/func_array_flip.asp

Tableaux

$tableau = array_filter($variable, "fonction")

retourne un tableau contenant les enregistrements filtrés d'un tableau à partir d'une fonction.

```
<?php
function impair($var)
{return ($var % 2 == 1);}
function pair($var)
{return ($var % 2 == 0);}
$array1 = array ("a"=>1, "b"=>2, "c"=>3, "d"=>4,
  "e"=>5);
$array2 = array (6, 7, 8, 9, 10, 11, 12);
echo "Impairs :\n";
print_r(array_filter($array1, "impair"));
echo "Pairs :\n";
print_r(array_filter($array2, "pair"));
?>
```

Tableaux

- Les fonctions de tableaux

$tableau = array_map($var_1 [, $var_2, ..., $var_N], 'fonction');

applique une fonction à un ou plusieurs tableaux.

$tableau = array_merge($var_1, $var_2, ..., $var_N);

enchaîne des tableaux entrés en argument afin d'en retourner un unique.

$tableau = array_merge_recursive($var_1, $var_2, ..., $var_N);

enchaîne des tableaux en conservant l'ordre des éléments dans le tableau résultant. Dans le cas de clés communes, les valeurs sont placées dans un tableau.

true | false = array_multisort($var, critère1, critère2 [, ..., $var_N, critère1, critère2])

trie un ou plusieurs tableaux selon un ordre croissant ou décroissant (SORT_ASC ou SORT_DESC) et selon une comparaison alphabétique, numérique ou de chaîne de caractères (SORT_REGULAR, SORT_NUMERIC ou SORT_STRING).

$tableau = array_pad($variable, taille, valeur);

recopie tout un tableau en ajustant sa taille à l'argument correspondant et en bourrant d'une valeur spécifiée les éléments vides.

PHP/Logique*Arithmétique

+	Addition	5 + 2
-	Soustraction	5 - 2
*	Multiplication	5 * 2
/	Division	5 / 2
%	Modulo	5%2

==	Egalité	$v == 5
<, <=	Infériorité	$v < 5
>, >=	Supériorité	$v >= 7
!=	Différent	$v != 7
%	Modulo	5%2

PHP/Opérateurs logiques et sur bits

&& and	et	$c1 && $c2 $c1 and $c2
|| or	ou	$c1 || $c2 $c1 or $c2
xor	ou exclusif	$c1 xor $c2
!	négation	!$c1
? :	Affectation conditionnelle (opérateur ternaire)	$c1 == 0 ? $c2 : $c3

&	et	$c1 & $c2
|	ou	$c1 | $c2
^	ou exclusif	$c1 ^ $c2
~	négation	~$c1
>> <<	Décalage à droite Décalage à gauche	$c1 <<2 $c2>>3

PHP Structures conditionnelles

- **if** (cond) { instructions 1; }
  elseif (cond) {instructions 2; }
  …
  else(cond) {instructions N; }
- **switch** (expression) {
  case val1: instruction; break;
  case val2: instruction; break;
  …
  default: instruction;
  }
- **for**(init, cond, evolution) {
  instructions;
  }
- **while** (condition) {
  instructions;
  [break;]
  }
- **do** {instructions}
  while (condition) ;

Inclusion de code et sortie

- Require ("fichier")
  - Recopie un fichier de code à sa place
  - Exemple : require ("opendb.php")
  - Erreur bloquante si fichier n'est pas trouvé. ➔ empeche le reste du code de s'exécuter
- Include ("fichier")
  - Idem require mais évalue et exécute le code à chaque rencontre de include ➔ include_once() pour éviter d'appeler le fichier en plusieurs reprises
  - Retourne une notification si le fichier n'est pas trouvé
  - Toutes les deux peuvent ramener aussi des codes CSS, HTML …
- Exit
  - Permet de quitter une page PHP
- Die ('message')
  - Die écrit le message et termine le programme

Arrêt prématuré d'un script

- <?php
  - if(file_exists("form.ph")
    - Require_once("form.ph")
  - else
    - exit(« Erreur de chargement du fichier »); // ou bien
    - die(« Erreur de chargement du fichier »);
- ?>

Les fonctions

- Bloc d'instructions avec paramètres optionnels créé par l'intermédiaire de l'instruction **function**

```
function nom_fonction([$param_1 [= val_déf], ..., $param_N [= val_déf]])
{
        Bloc d'instructions...
        [return valeurs;] }
```

- Appel de fonction de la forme :

```
$resultat = nom_fonction([arg_1, ..., arg_N]);
```

Portée des variables

```
function SurfaceTriangle($largeur, $hauteur) {
    $resultat = ($largeur * $hauteur) / 2;
    return $resultat; }

echo SurfaceTriangle(20, 10) .
" cm<sup>2</sup>"; // retourne 100 cm²

echo "($largeur * $hauteur) / 2 = " . $resultat; // instruction
erronée : variables indéfinies
```

Portée des variables

- Par défaut, toutes les variables sont locales
  - Leur portée se réduit à la fonction ou au bloc de leur déclaration
  - Pour déclarer une variable globale, on peut utiliser le tableau $_GLOBALS[ ]

```
<?php $_GLOBALS['MaVar']="Bonjour"; ?>
```

- Constantes

```
<?php
        define("USER","TOTO");
        echo USER; // Notez l'absence de $ ici
?>
```

Le mot clé GLOBAL

```
<?php
$a = 1; $b = 3;
    function somme($a,$b){
            global $a, $b;
            $b = $a+$b;
}
Somme($a,$b);
Echo $b;
```

Le script ci-dessus va afficher la valeur *4*. En déclarant globales les variables *$a* et *$b* locales de la fonction somme(), toutes les références à ces variables concerneront les variables globales. Il n'y a aucune limite au nombre de variables globales qui peuvent être manipulées par une fonction

$GLOBALS

- Le tableau *$GLOBALS* est un tableau associatif avec le nom des variables globales comme clé et les valeurs des éléments du tableau comme valeur des variables. Notez que *$GLOBALS* existe dans tous les contextes, car *$GLOBALS* est un superglobal. Voici un exemple des super globaux :

```
<?php
function test() {
    $foo = "variable locale";

    echo '$foo dans le contexte global : '. $GLOBALS["foo"]."\n";
    echo '$foo dans le contexte courant : '.$foo."\n";
}
$foo = "Exemple de contenu";
test();
?>
```

Exemple montrant les superglobales et la portée

- **Note**:
  - L'utilisation du mot clé *global* à l'extérieur d'une fonction n'est pas une erreur. Il peut être utilisé si le fichier est inclus depuis l'intérieur d'une fonction.

```php
<?php
function test_superglobal()
{
    echo $_POST['name'];
}
?>
```

Notez que *$HTTP_POST_VARS* et *$_POST* sont des variables différentes et que PHP les traite comme telles

Utilisation des variables *static*

- Une autre caractéristique importante de la portée des variables est la notion de variable *static*. Une variable statique a une portée locale uniquement, mais elle ne perd pas sa valeur lorsque le script appelle la fonction. Prenons l'exemple suivant :

A

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

B

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

Explication du diapo précèdent

- A: test est un peu inutile car à chaque fois qu'elle est appelée, elle initialise *$a* à *0* et affiche "*0*". L'incrémentation de la variable (*$a*++) ne sert pas à grand chose, car dès que test est terminée, la variable *$a* disparaît. Pour faire une fonction de comptage utile, c'est-à-dire qui ne perdra pas la trace du compteur, la variable *$a* est déclarée comme une variable statique :
- B: Maintenant, la variable *$a* est initialisée uniquement lors du premier appel à la fonction et, à chaque fois que *test()* est appelée, elle affichera une valeur de *$a* incrémentée de 1.

Static et la récursivité

- Les variables statiques sont essentielles lorsque vous faites des appels récursifs à une fonction. Une fonction récursive est une fonction qui s'appelle elle-même. Il faut faire attention lorsque vous écrivez une fonction récursive car il est facile de faire une boucle infinie. Vous devez vérifier que vous avez bien une condition qui permet de terminer votre récursivité. La fonction suivante compte récursivement jusqu'à 10, en utilisant la variable *$count* pour savoir quand il faut s'arrêter :

```
<?php
function test()
{
    static $count = 0;
    $count++;
    echo $count;
    if ($count < 10) {  test();
    }
    $count--;
}  ?>
```


```
<?php
function test(){
$count = 0;
$count++;
echo $count;
if($count < 10){

test();}
$count--;
}

test();
?>
```

Static et la récursivité

- **Note**:
  - Les variables statiques doivent être déclarées comme dans l'exemple ci-dessus. À partir de PHP 5.6 vous pouvez affecter des valeurs à ces variables qui sont le résultat d'expressions, **mais vous ne pouvez pas utiliser une fonction ici**, ce qui va provoquer une erreur d'analyse.

```
<?php
function foo(){
    static $int = 0;          // correct
    static $int = 1+2;        // correct  (depuis PHP 5.6)
    static $int = sqrt(121);  // faux  (car c'est une fonction)
    $int++;
    echo $int;
}
?>
```

Exemples : fonctions sur variables

- gettype($var)
  - retourne le type de la variable argument
- settype($var)
  - change le type d'une variable, ce qui peut impliquer une conversion
- isset($var) / empty($var)
  - permet de tester si une valeur a été affectée à la variable / ou ne l'a pas été
- unset($var)
  - détruit une variable
- is_int/is_double/is_string
  - Permettent de tester le type d'une variable

Exemple

```
<?php
    // ceci est un commentaire   // variable utilisée sans avoir été déclarée
    $nom="dupont";
    print "nom=$nom\n";     // un affichage écran
    // un tableau avec des éléments de type différent
    $tableau=array("un","deux",3,4);
    $n=count($tableau);      // son nombre d'éléments
    for($i=0;$i<$n;$i++)
          print "tableau[$i]=$tableau[$i]\n";
    // initialisation de 2 variables avec le contenu d'un tableau
    list($chaine1,$chaine2)=array("chaine1","chaine2");
    // concaténation des 2 chaînes
    $chaine3=$chaine1.$chaine2;
    // affichage résultat
    print "[$chaine1,$chaine2,$chaine3]\n";
```

Exemple

```
affiche($chaine1); // utilisation fonction
afficheType($n); // le type d'une variable peut être connu
afficheType($chaine1); // fonction à créer
afficheType($tableau);
$n="a changé"; // le type d'une variable peut changer au cours d'exécution
afficheType($n);
$res1=f1(4); // une fonction peut rendre un résultat
print "res1=$res1\n";
// une fonction peut rendre un tableau de valeurs
list($res1,$res2,$res3)=f2();
print "(res1,res2,res3)=[$res1,$res2,$res3]\n";
$t=f2(); // on aurait pu récupérer ces valeurs dans un tableau
for($i=0;$i<count($t);$i++)
    print "t[$i]=$t[$i]\n";
```

Exemple

```
for($i=0;$i<count($t);$i++)     // des tests
    if (getType($t[$i])=="string")    // n'affiche que les chaînes
            print "t[$i]=$t[$i]\n";
for($i=0;$i<count($t);$i++) // d'autres tests
     if (getType($t[$i])=="integer" and $t[$i]>10) // n'affiche que les entiers >10
            print "t[$i]=$t[$i]\n";
$t=array(8,5,0,-2,3,4); // une boucle while
$i=0;
$somme=0;
while($i<count($t) and $t[$i]>0){
            print "t[$i]=$t[$i]\n";
            $somme+=$t[$i]; //$somme=$somme+$t[$i]
            $i++; //$i=$i+1
}//while
print "somme=$somme\n";
exit; // fin programme
?>
```

Les chaînes en PHP

- Guillemets ou Cotes :

```
<?php
    $var="Hello PHP";
    $machaine="le contenu de \$var est $var<br>";
    echo $machaine;
   //ou avec des '  ':
   $mystring='le contenu de $var est '.$var;
   echo $mystring;
?>
```

- La concaténation : A l'aide de .
- La longueur d'une chaine : <?php int lg=strlen($chaine); ?>
- Exemple: <?php int lg=strlen($var); ?>

Chaines en PHP

- Accéder au caractère i d'une chaine :

  <?php **echo** $chaine[i]; ?>

  La chaine est traitée comme un tableau indexé par un entier

  La plupart des tableaux de PHP sont indexés par des chaines...

- Mettre en majuscules/minuscules :
  - avec strtoupper() pour obtenir des majuscules
  - avec strtolower() pour mettre en minuscules
  - avec ucfirst() pour mettre en majuscule la première lettre d'une chaine
  - avec ucwords() pour mettre en majuscule la première lettre de chaque mot dans une chaine
- Cherche de sous-chaines ou de motifs dans une chaine
  - avec strstr()
  - avec stristr()
  - avec pregmach

strstr()

- <?php
  echo strstr("Hello world!","world");
  ?>
- The strstr() function searches for the first occurrence of a string inside another string.
  - **Note:** This function is binary-safe.
  - **Note:** This function is case-sensitive. For a case-insensitive search, use stristr() function.
- **Syntax**

  strstr(*string,search,before_search*)

strstr()

Parameter	Description
string	Required. Specifies the string to search
search	Required. Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number
before_search	Optional. A boolean value whose default is "false". If set to "true", it returns the part of the string before the first occurrence of the *search* parameter.

- Technical details:

Return Value:	Returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.
PHP Version:	4+
Changelog:	The *before_search* parameter was added in PHP 5.3

- Example
  - Search a string for the ASCII value of "o" and return the rest of the string:

```
<?php    echo strstr("Hello world!",111);     ?>
```

- Example
  - Return the part of the string before the first occurence of "world":

```
<?php echo strstr("Hello world!","world",true);   ?>
```

Expressions régulières - Les regex POSIX et PCRE

- Une expression régulière est une structure logique connue aussi sous le nom de "model" ou "pattern" qui permet de décrire la forme que doit vérifier une chaîne de caractères. Imaginez qu'on invite un client à saisir une adresse mail dans un champ de saisie. Afin de s'assurer si ce qu'il a entré respecte la forme conventionnelle d'un email, on peut vérifier les points suivants:
- Est que l'arobas figure-t-il dans l'adresse mail?
- Y a-t-il un point avant le TLD?
- Est ce qu'il y a un caractère ou plus avant l'arobas?
- Le nom de domaine semble-t-il valide?
- Est ce que l'email ne contient pas de caractères non autorisés comme les espaces?
-

POSIX et PCRE

- En PHP, il existe deux familles d’expressions régulières. Elle font toutes la même chose: chercher des motifs au sein d'une chaîne de caractères. Bien qu'il partagent des fois les mêmes caractères spéciaux au sein des models, il existe quand même de nettes différences entre les deux. Ces deux familles sont: **POSIX**: il s'agit d'une famille d'expressions régulières utilisée en PHP. POSIX est l'acronyme de Portable Operating System Interface.
- **PCRE**: signifie Perl Compatible Regular Expressions. C'est une famille d'expression régulières qui vient du langage PERL.
- La principale différence entre les deux familles c'est le temps d’exécution. En effet, les PCRE sont plus rapides que les POSIX, surtout quand il s'agit d'expressions régulières longues et complexes. Une autre différence se manifeste dans la syntaxe. Bien que la plupart des caractères spéciaux des expressions peuvent être utilisés pour les deux familles. Il en existe qui sont propres à chaque famille.

Expressions régulières POSIX

- **Les fonctions**
  - Les fonctions qui traitent les expressions régulières de la famille POSIX les plus courantes sont: ereg(), mb_ereg(), ereg_replace(), mb_ereg_replace(), split() et mb_split().

- **Fonctions ereg() et eregi():**
  - La fonction **ereg($motif,$chaine)** permet de vérifier l'existence du motif $motif dans la chaîne de caractères $chaine. La variable $motif représente l'expression régulière ou le model à vérifier. La fonction ereg() retourne **true** si le motif existe dans la chaîne ou **false** si le motif n'est pas trouvé.

- Exemple

```
<?php
      $str="Bonjour";
      if(ereg("^B",$str))
           echo "La chaîne commence par B";
       else
            echo "La chaîne ne commence pas par B";
?>
```

- **Fonctions mb_ereg() et mb_eregi():**
  - La fonction **mb_eregi($motif,$chaine)** fait le même traitement que la fonction **ereg()**. Cependant, elle permet de supporter les expressions régulières qui font appel à des caractères multi-octés (multi byte).
  - La fonction **mb_eregi()** ne vérifie pas la casse.
- **Fonctions ereg_replace() et eregi_replace():**
  - La fonction **ereg_replace($motif,$remp,$chaine)** permet de chercher l'expression $motif dans la chaîne de caractères $chaine et le remplacer par $remp.
  - Exemple:

```
<?php
  $str="Bonjour";
  echo ereg_replace("jour$","soir",$str);
?>
```

Fonctions mb_ereg_replace() et mb_eregi_replace()

- La fonction **mb_ereg_replace($motif,$remp,$chaine)** fait le même traitement que la fonction ereg_replace() avec le support des caractères multi-octets.
- La fonction **mb_eregi_replace()** ne vérifie pas la casse.

- **Fonctions split() et spliti()**:

La fonction **split($motif,$chaine)** permet de découper la chaîne de caractères au niveau des occurrences $motif. Les morceaux de la chaîne seront ensuite stockés dans un tableau. Son fonctionnement rappelle celui de la fonction **explode()**. Cependant, split() prend en charge les expressions régulières.

Exemple:

```
•<?php
   $str="Bonjour";
   $tab=split("o",$str);
   print_r($tab);
?>
```

Caractères spéciaux d'une expression régulière

Caractères spécial	Signification
.	Un seul caractère quelconque
+	Le motif précédent doit figurer au moins une fois, le max n'est pas déterminé
*	Le motif précédent doit figurer au moins 0 fois, le max n'est pas déterminé
()	Pour grouper un ensemble de caractères en tant que motif unique
[]	Pour désigner un ensemble de motifs dont , au moins, l'un d'entre eux doit figurer
-	Désigne un intervalle s'il est déclaré entre les crochets. Exemple: [a-z] signifie l'alphabet minuscule
?	Le motif précédent doit figurer 0 ou une fois.
^	Le motif suivant doit figurer au début de la chaîne
$	Le motif précédent doit figurer à la fin de la chaîne
{}	Pour spécifier le minimum et le maximum de fois où l'occurrence doit figurer. Exemple: {1,3} désigne que l'occurrence figure entre 1 et 3 fois.

Séquences utiles pour simplifier les expressions régulières POSIX

Par exemple si on veut vérifier la présente d'un caractère alphabétique (minuscule ou majuscule) ou caractère numérique au sein d'une chaîne de caractères. Le model contiendra donc quelque chose qui ressemble à **[a-zA-Z0-9]**. Heureusement, il existe des séquences pratiques pour les expressions régulières de la famille POSIX. Par exemple, le model précédent peut s'écrire **[[:alnum:]]** . Voici la liste des séquences les plus fréquentes

Séquence	Signification
[[:alnum:]]	Tous les caractères alphabétiques ou numériques (équivalent à [a-zA-Z0-9])
[[:alpha:]]	Tous les caractères alphabétiques (équivalent à [a-zA-Z])
[[:digit:]]	Tous les caractères numériques (équivalent à [0-9])
[[:blank:]]	Tous les types d'espaces et tabulation
[[:xdigit:]]	Caractères hexadécimaux (équivalent à [a-fA-F0-9])
[[:graph:]]	Tous les caractères affichables et imprimables
[[:punct:]]	Tous les caractères de ponctuation
[[:cntrl:]]	Caractères d’échappement

Expressions régulières PCRE

Il s'agit de la famille des expressions régulières la plus recommandée vu sa rapidité surtout lorsqu'il s'agit de motifs complexes.

- **Les fonctions**

Nous allons nous intéresser aux fonctions preg_match(), preg_replace() et preg_split().

- **Fonction preg_match():**
  - La fonction **preg_match($motif,$chaine)** peut contenir plusieurs paramètres. Pour simplifier, nous allons nous contenter des deux premiers à savoir, $motif qui désigne l'expression régulière et $chaine qui représente la chaîne de caractères dans laquelle on vérifie la présence du motif.
  - Le motif d'une expression PCRE doit toujours être placé entre des délimiteurs de votre choix. Le plus courant c'est d'utiliser un slah ou un dièse.

- Par exemple, si on veut vérifier la présence de l'arobas dans la chaîne $email, alors le code ressemblerait à ceci:

```
<?php
  $email="user@domaine.tld";
  if(preg_match("#@#",$email))
    echo "L'arobas existe.";
  else
    echo "L'arobas n'existe pas.";
?>
```

- Vous avez remarqué que nous avons placé le motif **@** entre deux dièses dans ce cas.
  Imaginons maintenant qu'on veut vérifier si un Email est valide, alors on peut faire ceci:

```
<?php
  $email="user@domaine.tld";
  if(preg_match("#^[a-zA-Z0-9-_.]+@[a-zA-Z0-9-.]+.[a-zA-Z]{2,6}$#",$email))
    echo "Email valide.";
  else
    echo "Email invalide.";
?>
```

Fonction preg_replace

- La fonction **preg_replace($motif,$remp,$chaine)** permet de chercher le motif $motif dans la chaîne de caractères $chaine et le remplacer par $remp.

  Exemple:

```
<?php
   $email="user@domaine.tld";
   echo preg_replace("#.tld$#",".com",$email);
?>
```

Fonction preg_split

- La fonction **preg_split($motif,$chaine)** éclate la chaîne de caractères $chaine au niveau des occurrences décrits par $motif et dépose les fragments de la chaîne dans un tableau.
  Exemple:

```
<?php
  $str="Bonjour";
  $tab=preg_split("#o#",$str);
  print_r($tab);
?>
```

Séquences utiles pour simplifier les expressions régulières PCRE

- Comme pour la famille POSIX, PCRE dispose d'un ensemble de séquences qui permettent de simplifier nettement l'écriture des expressions régulières.
Ce tableau donne un aperçu sur les séquences les plus courantes:

Séquence	Signification
d	Tous les chiffres (équivalent à [0-9])
D	Tout ce qui n'est pas un chiffre (équivalent à [^0-9])
w	Tous les caractères alphabétique, numériques ou caractère sous-tiret (équivalent à [a-zA-Z0-9_])
W	Tous les caractères qui ne sont pas alphabétiques, numériques ou caractère sous-tiret (équivalent à [^a-zA-Z0-9_])
t	Tabulation
s	Espace
r	Retour chariot
n	Nouvelle ligne

Recap: Regular expression

- A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of text search and text replace operations.
- **Syntax**

In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

```
$exp = "/smis3fpk/i";
```

In the example above, / is the **delimiter**, *smis3fpk* is the **pattern** that is being searched for, and i is a **modifier** that makes the search case-insensitive.
The delimiter can be any character that is not a letter, number, backslash or space. The most common delimiter is the forward slash (/), but when your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

Recap: Regular Expression Functions

PHP provides a variety of functions that allow you to use regular expressions. The preg_match(), preg_match_all() and preg_replace() functions are some of the most commonly used ones:

Function	Description
preg_match()	Returns 1 if the pattern was found in the string and 0 if not
preg_match_all()	Returns the number of times the pattern was found in the string, which may also be 0
preg_replace()	Returns a new string where matched patterns have been replaced with another string

Recap: preg_match

- La syntaxe : **preg_match (motif de chaîne , chaîne sujet *[, tableau résultat]*);**

  Ce qui est en italique est optionnel
- preg_match retourne vrai si la chaîne sujet contient le motif donné. Si l'option *tableau résultat* est fournie, preg_match remplit un tableau avec les correspondances.
- $tableau_resultat, il contiendra les résultats de la recherche : $tableau_resultat[0] contiendra la partie qui satisfait le masque complet.
- $tableau_resultat[1] contiendra la partie qui satisfait la première parenthèse capturante et ainsi de suite...

Exemple

```
<?php
$str = “FP Khouribga";
$pattern = "/Khouribga/i";
echo preg_match($pattern, $str); // Outputs 1
?>
```

Recap: preg_match

- Exemple:
  - Test sur un champ login lettres chiffres uniquement
    Niveau de compréhension : facile

```
$login="Toto1508";
if(!preg_match('/^[[:alnum:]]{4,8}$/',$login))
{
echo"Le login n'est pas correct";
}
else
{
echo"Le login est correct";
}
?>
```

  - Explication :
    Le login ne peut contenir que des lettres et des chiffres et doit être de 4 caractères minimum à 8 caractères maximum.
    [[:alnum:]] classe déterminant l'usage exclusif des lettres et chiffres.
    {4,8} est l'intervalle de reconnaissance imposant 4 caractères minimum et 8 maximum.

Explications

- [] => character class definition
- ^ => negate the class
- a-z => chars from 'a' to 'z‘
- _ => underscore
- => hyphen '-' (You need to escape it)
- 0-9 => numbers (from zero to nine)
- The 'i' modifier at the end of the regex is for 'case-insensitive' if you don't put that you will need to add the upper case characters in the code before by doing A-Z

Using preg_match_all()

The preg_match_all function will tell you how many matches were found for a pattern in a string.

```
<?php
$str = "The rain in SPAIN falls mainly on the plains.";
$pattern = "/ain/i";
echo preg_match_all($pattern, $str); // Outputs 4
?>
```

Using preg_replace()

- This function will replace all of the matches of the pattern in a string with another string.

```
<?php
$str = "Visit Microsoft!";
$pattern = "/microsoft/i";
echo preg_replace($pattern, "FPK", $str); // Outputs "Visit FPK!"
?>
```

- **Exemple 2**
  - Idem que le précédent **plus (_)** underscore en utilisant la classe prédéfinie **\w**

```
$login="Toto_1508";
if(!preg_match('/^\w{4,8}$/',$login))
{
   echo"Le login n'est pas correct";
}
else
{
   echo"Le login est correct";
}
?>
```

Exemple à faire chez soit

- **Exemple 3**
  - Test sur un email
    Niveau de compréhension :difficile

```
preg_match('/^[[:alnum:]]([-_.]?[[:alnum:]])*@[[:alnum:]]
([-_.]?[[:alnum:]])*\.([a-z]{2,4})$/',$chaine);

//--- exemple avec les classes prédéfinies

preg_match('/^\w([-_.]?\w)*@\w([-_.]?\w)*\.([a-
z]{2,4})$/',$chaine)

?>
```

Explications

L'adresse email doit absolument commencer par une lettre ou un chiffre **[[:alnum:]]** puis peut être suivie d'un seul - ou d'un seul _ ou encore d'un seul point, suivi d'une série de lettres ou chiffres **([-_.]?[[:alnum:]])**

Vient ensuite le @ suivi uniquement par une ou plusieurs lettres ou chiffres **[[:alnum:]]**, puis peut être suivie d'un seul - ou d'un seul _ ou encore d'un seul point, suivi d'une série de lettres ou chiffres **([-_.]?[[:alnum:]])***

Vient enfin le point suivi par 2 à 4 lettres pour le domaine **\.([a-z]{2,4}**

Vous remarquerez le \ devant le point qui permet de l'échapper afin qu'il ne soit plus considéré comme un méta caractère mais bien comme le signe . (point)

ctype_alnum() (Check for Alphanumeric)

- A ctype_alnum() function in PHP used to check all characters of given string/text are alphanumeric or not. If all characters are alphanumeric then return TRUE, otherwise return FALSE.

Syntax:

bool ctype_alnum ($text)

- **Parameters Used:**

$text : It is a mandatory parameter which specifies the string.

Example 1:

```
<?php
// PHP program to check given string is
// all characters are alphanumeric
$string = ‘FPKhouribga';
    if ( ctype_alnum($string))
        echo "Yes\n";
    else
        echo "No\n";
?>
```

Exemple 2

```
<?php
// PHP program to check given string is  all characters are alphanumeric
$FPK_chaine = array(
    ‘Fpkhouribga',
    ‘fpk@gmail.com',
    '2021',
    ‘fpk2021',
    ‘F P K ',
    '@!$%^&*()|2021'
);

// Checking above given four strings by use of  ctype_alnum() function .
foreach ($FPK_chaine as $pars) {

    if (ctype_alnum($pars))
        echo “Ok\n";
    else
        echo “Ko\n";

}

?>
```

Regular Expression Modifiers

- Modifiers can change how a search is performed.

Modifier	Description
i	Performs a case-insensitive search
m	Performs a multiline search (patterns that search for the beginning or end of a string will match the beginning or end of each line)
u	Enables correct matching of UTF-8 encoded patterns

Regular Expression Patterns

- Brackets are used to find a range of characters:

Expression	Description
[abc]	Find one character from the options between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

Metacharacters

- Metacharacters are characters with a special meaning:

Metacharacter	Description
|	Find a match for any one of the patterns separated by | as in: cat|dog|fish
.	Find just one instance of any character
^	Finds a match as the beginning of a string as in: ^Hello
$	Finds a match at the end of the string as in: World$
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiers

- Quantifiers define quantities:

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{x}	Matches any string that contains a sequence of *X n*‘s
n{x,y}	Matches any string that contains a sequence of X to Y *n’s*
n{x,}	Matches any string that contains a sequence of at least X *n*‘’s

Exemples

Grouping

- You can use parentheses () to apply quantifiers to entire patterns. They also can be used to select parts of the pattern to be used as a match.
- Example:
- Use grouping to search for the word "banana" by looking for *ba* followed by two instances of *na*:
- ```
  <?php
  $str = "Apples and bananas.";
  $pattern = "/ba(na){2}/i";
  echo preg_match($pattern, $str); // Outputs 1
  ?>
  ```
  ```

Regex

- Pour plus de références sur regex :
- https://www.w3schools.com/php/php_regex.asp
- https://www.w3schools.com/php/php_ref_regex.asp

Fonctions de test

- is_int()
- is_long()
- is_double()
- is_array()
- is_object()
- is_string()

Attention : N'oubliez pas comme en JavaScript la différence entre l'opérateur == et ===

Le premier vérifie l'égalité des contenus en ne tenant pas compte d'une éventuelle différence de typage (int ou string par exemple) tandis que le second vérifie une égalité stricte.

En d'autres termes : 5 == « 5 » est VRAI tandis que 5 === « 5 » est FAUX

each() Function

- The each() function is an inbuilt function in PHP and is used to get the current element key-value pair of the given array to which the internal pointer is currently pointing. After returning the key and value of the current element the internal pointer is incremented by one in the array.
- **Note**: You can use reset() function if you want to traverse the array again using each().

Syntax:

```
each($array)
```

Each() example

1)

```
<?php
$arr = array(‘S1', ‘S2', ‘S3');
while($v = each($arr)){
echo($v['key']." ====>".$v['value']."<br>");
}?>
```

2)

```
<?php
$arr = array(‘S4', 'S5', ‘S6');
while(list($k,$v) = each($arr)){
echo "$k  ======= > $v <br>";
}?>
```

Date()

- La fonction **date()** permet de retourner une chaîne de caractères ou une valeur booléenne (selon les paramètres qui lui sont passés). Dans le cas général, la fonction **date()** retourne des informations relatives à la date et l'heure courante. Cependant on peut toujours spécifier une date antérieure ou postérieure.
- La fonction **date(**format[,timestamp]**)** accepte deux paramètres:
  - **Format**: il s'agit d'une chaîne de caractères qui dicte le format à suivre pour afficher les informations relatives à la date et l'heure par le biais de caractères prédéfinis.
  - **Timestamp**: c'est un entier qui sert à indiquer la date et l'heure pour lesquelles on souhaite retourner les informations décrites par le format. Ce paramètre est facultatif. S'il n'est pas spécifié alors la fonction **date()** retournera les informations relatives à la date courante.

Date()

- Exemple:
  - echo date("Y"); // Affiche 2021 (au moment de l'exécution)
  - echo date("d/m/Y"); // Affiche 06/11/2021 (au moment de l'exécution)
- La fonction **date()** évalue le premier paramètre (format). Elle remplace les caractères prédéfinis par leurs valeurs et affiche les autres caractères tels qu'ils sont. La deuxième instruction du code précédent contient des slashs qui sont affichés tels qu'ils sont. Par contre, les autres caractères ont été remplacés par leur valeurs.

Autres Fonctions PHP(3)

- **Les formats de date et d'heure**

  a représente am (matin) ou pm (après-midi).
  A représente AM (matin) ou PM (après-midi).
  B représente une heure Internet Swatch.
  d représente le jour du mois sur deux chiffres allant de 01 à 31.
  D représente le jour de la semaine en trois lettres et en anglais (Sun, ..., Sat).
  F représente le mois complet en anglais (January, ..., December).
  g représente une heure au format 12 heures allant de 1 à 12.
  G représente une heure au format 24 heures allant de 1 à 24.
  h représente une heure au format 12 heures avec un zéro de complément allant de 00 à 11.
  H représente une heure au format 24 heures allant de 00 à 23.
  i représente les minutes allant de 00 à 59.
  I est égal à 1 si l'heure d'été est activée ou 0 pour l'heure d'hiver.
  j représente le jour du mois allant de 1 à 31.
  l représente le jour de la semaine complet et en anglais (Sunday, ..., Saturday).
  L est égal à 1 si l'année est bissextile, sinon 0.
  m représente un mois allant de 01 à 12.
  M représente un mois en trois lettres et en anglais (Jan, ..., Dec).

Autres Fonctions PHP(4)

- **Les formats de date et d'heur**

n	représente un mois allant de 1 à 12.
O	représente la différence d'heures avec l'heure de Greenwich (+0100).
r	représente un format de date conforme au RFC 822 (Mon, 25 Mar 2002 05:08:26 +0100).
s	représente les secondes allant de 00 à 59.
S	représente le suffixe ordinal d'un nombre en anglais et sur deux lettres th ou nd.
t	représente le nombre de jours dans le mois (28, 29, 30 ou 31).
T	représente le fuseau horaire.
U	représente les secondes depuis une époque.
w	représente le jour de la semaine allant de 0 (Dimanche) à 6 (Samedi).
Y	représente une année sur quatre chiffres (2002).
y	représente une année sur 2 chiffres (02).
z	représente le jour de l'année allant de 0 à 365.
Z	représente le décalage horaire en secondes.

Exemple

- <?php
  echo date("d/m/Y"); // Affiche la date du jour
  echo "Il est " . date("H:i:s") ; // Affiche l'heure
  ?>

Le timestamp PHP

- Un timestamp, c'est un nombre.
- C'est le nombre de secondes écoulées depuis le 1er Janvier 1970 à Minuit.
- Pourquoi depuis le 1er Janvier 1970 à Minuit ? C'est symbolique, il fallait bien prendre un point de départ.
- En fait, ça représente le début de l'époque où le système d'exploitation Unix a été créé.
- Le 1er Janvier 1970 à Minuit, le timestamp avait pour valeur 0. Aujourd'hui, beaucoup beaucoup de secondes se sont écoulées
- Pour afficher le timestamp de la seconde actuelle, on utilise la fonction
- **time()**

Exemple

- <?php
  echo time(); //Affiche le nbre de sec écoulées depuis le 1er janvier 1970
  ?>

- Le timestamp avec la fonction date
- Il est possible de fournir un second paramètre à date (après les lettres) : le timestamp sur lequel vous voulez obtenir des informations.
- Par défaut, date utilise le timestamp actuel : elle renvoie donc l'heure actuelle, le jour actuel etc...
  Mais si vous lui donnez un timestamp, elle fera des calculs sur ce moment-là.
- Pour faire un test grandeur nature, je vous donne le timestamp qu'il était au moment où j'ai écrit ces lignes
  - <?php
    $timestamp1=1261774471; // Timestamp affiché quand j'ai écris ce cours
    echo date('d/m/Y',$timestamp1);
    ?>

Fonction checkdate()

- La fonction **checkdate()** permet de vérifier la validité d'une date. Si la date passée en paramètre est valide alors elle retourne true, sinon elle retourne false.
- La syntaxe de la fonction **checkdate()** vérifie cette forme: **checkdate(mois , jour , année)**

```
<?php
   if(checkdate(30,02,2021))
      echo "Date valide.";
   else
      echo "Date invalide.";
?>
```

Fonction time()

- La fonction **time()** calcule le timestamp Unix en secondes. Cela représente le nombre de secondes écoulées depuis le 1 janvier 1970 à minuit jusqu'à l'instant actuelle. Cette date est prise comme référence car elle correspond à la date du lancement du premier système Unix. Certains préfèrent l'appeler Timestamp POSIX.
- Exemple:

```
<?php
   echo time();
?>
```

Fonction mktime()

- La fonction **mktime()** calcule le timestamp Unix en secondes à partir du 1 janvier 1970 à minuit jusqu'à la date et l'heure spécifiés en paramètre.
- La syntaxe de la fonction **mktime()** vérifie cette forme:
- **mktime(heure , minute , seconde , mois , jour , année)**
- Exemple:

```
<?php
   echo mktime(00,00,00,01,01,2016);
?>
<?PHP
$demain = mktime(0,0,0,date("m"),date("d")+1,date("Y"));
echo "demain sera le : " .date ("d/m/Y", $demain) ; ?>
```

Manipuler une date antérieure ou postérieure avec la fonction date()

- Jusqu'ici, nous avons vu comment extraire des informations sur la date et l'heure actuelle à l'aide de la fonction **date()** en manipulant son premier paramètre (format). Il est maintenant temps d'explorer son deuxième paramètre (timestamp).
- Supposons qu'on veut connaitre le jour (en anglais) qui correspond à la date 01/01/2020. Dans ce cas il faut placer le timestamp relatif à cette date en guise de deuxième paramètre de la fonction **date()**.
- Le code ressemblera donc à ceci:

```
<?php
   echo date("l",mktime(00,00,00,01,01,2016));
?>
```

setlocale() et strftime()

Par défaut, les dates vont être renvoyées en anglais par la plupart des serveurs.
Pour transformer une date en anglais vers du français, nous avons plusieurs solutions mais une est recommandée : l'utilisation des fonctions setlocale() et strftime().
La fonction setlocale() va nous permettre de modifier et de définir de nouvelles informations de localisation. On va déjà pouvoir passer une constante à cette fonction qui va définir les données qui doivent être définies localement : la comparaison de chaines de caractères, la monnaie, les chiffres et le séparateur décimal, les dates ou tout cela à la fois.

Attention cependant : les caractères ne vont pas forcément être les mêmes et signifier la même chose que pour date() et nous allons cette fois-ci devoir les préfixer avec un %.

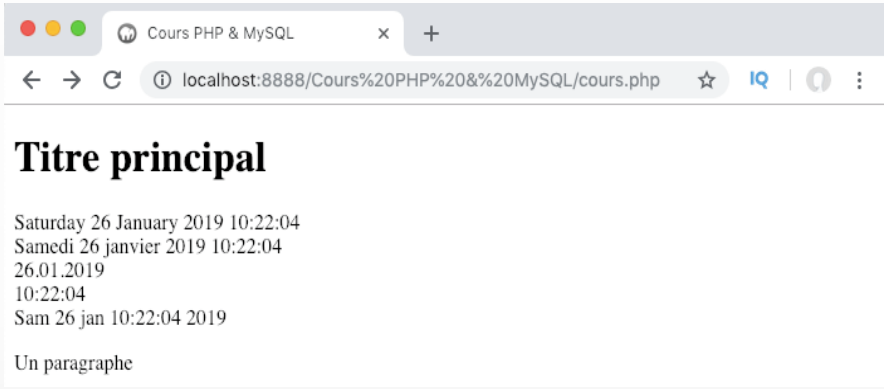
Caractère	Signification
%a	Représente le jour de la semaine en trois lettres en anglais
%A	Représente le jour de la semaine en toutes lettres en anglais
%u	Représente le jour de la semaine en chiffre au format ISO-8601 (lundi 1, dimanche = 7)
%w	Représente le jour de la semaine en chiffre (dimanche = 0, samedi = 6)

%d	Représente le jour du mois en deux chiffres (entre 01 et 31)
%j	Représente le jour de l'année avec les zéros de 001 à 366
%U	Représente le numéro de la semaine de l'année en ne comptant que les semaines pleines
%V	Représente le numéro de la semaine de l'année en suivant la norme ISO-8601 (si au moins 4 jours d'une semaine se situent dans l'année alors la semaine compte)
%m	Représente le mois sur deux chiffres de 01 à 12
%b	Représente le nom du mois en lettres en abrégé
%B	Représente le nom complet du mois
%y	Représente l'année sur deux chiffres
%Y	Représente l'année sur 4 chiffres

%H	Représente l'heure, de 00 à 23
%k	Représente l'heure de 0 à 23
%I (i majuscule)	Représente l'heure de 01 à 12
%M	Représente les minutes de 00 à 59
%S	Représente les secondes de 00 à 59
%T	Identique à %H:%M:%S
%D	Identique à %m/%d/%y
%x	Représente la date sans l'heure au format préféré en se basant sur la constant locale
%c	Affiche la date et l'heure basées sur la constant locale

Exemple

```
<!DOCTYPE html>
<html>
<head>
<title>Cours PHP & MySQL</title>
<meta charset="utf-8">
<link rel="stylesheet" href="cours.css">
</head>
<body>
<h1>Titre principal</h1>
<?php
echo strftime('%A %d %B %Y %I:%M:%S'). '<br>';
setlocale(LC_TIME, ['fr', 'fra', 'fr_FR']);
echo strftime('%A %d %B %Y %I:%M:%S'). '<br>';
echo strftime('%x'). '<br>';
echo strftime('%T'). '<br>';
echo strftime('%c'). '<br>';
?>
<p>Un paragraphe</p>
</body>
</html>
```



Autres Fonctions PHP: Les dates et les heures

- **Les fonctions de date et d'heure**

true | false = checkdate(mois, jour, année);

vérifie la validité d'une date.

$chaine = date(format [, nombre]);

retourne une chaîne de caractères date/heure selon le format spécifié et représentant la date courante par défaut.

$tableau = getdate([nombre]);

retourne les éléments de date et d'heure dans un tableau associatif.

$tableau = gettimeofday();

retourne l'heure courante dans un tableau associatif.

$chaine = gmdate(format [, nombre]);

retourne une chaîne de caractères date/heure GMT/CUT selon le format spécifié et représentant la date courante par défaut.

$nombre = gmmktime(heure, minute, seconde, mois, jour, année [, 1/0]);

retourne l'instant UNIX d'une date GMT spécifiée et avec éventuellement une heure d'hiver

Autres Fonctions PHP(2)

- **Les fonctions de date et d'heure**

  $chaine = gmstrftime(format [, nombre]);

  formate une date/heure GMT/CUT en fonction des paramétrages locaux définis par setlocale.

  $tableau = localtime([nombre][, tab_associatif]);

  retourne l'heure locale dans un tableau indicé par défaut ou associatif (1)

  $chaine = microtime();

  retourne l'instant UNIX courant en secondes et microsecondes (1 janvier 1970 à 0H00)

  $nombre = mktime(heure, minute, seconde, mois, jour, année [, 1/0]);

  retourne l'instant UNIX d'une date spécifiée et avec éventuellement une heure d'hiver (1)

  $chaine = strftime(format [, instant]);

  formate une date/heure locale avec les options locales

  $nombre = time();

  retourne l'instant UNIX courant

date_diff()

- Permet de faire un calcul entre 2 dates.
  - <?php
    $datetime1 = date_create($date);
    $datetime2 = date_create($date_terminee);
    $interval = date_diff($datetime1, $datetime2);

    echo $interval->format('%R%a days');
    ?>

extension Intl pour formats plus riches

- <?php
- # Avec l'extension intl : echo "Avec l'extension Intl :";
- $dt = new DateTime;
- $formatter = new IntlDateFormatter('fr_FR', IntlDateFormatter::SHORT, IntlDateFormatter::SHORT);
- $formatter->setPattern('E d.M.yyyy');

- echo $formatter->format($dt).PHP_EOL;
- Résultat:
- Avec l'extension Intl :jeu. 7.10.2021

Les collections en PHP

- **En PHP standard, Collections = Arrays :**
  - Si on se contente de ce qu'offre PHP en standard, les collections se résument à l'utilisation des tableaux associatifs en PHP
  - Le framework des *Collections* en Java est beaucoup plus riche
- **DataStructures ds:**

  Il faut installer une librairie supplémentaire *ds* (DataStructures) pour avoir accèder à un Framework similaire en PHP.
  On utilise la commande *pecl* pour installer *ds* comme une extension:
  pecl install ds
  Puis charger l'extension en ajoutant aux fichiers .ini de PHP:
  extension=ds.so

- Cette extension nous donne accès à des classes similaires à celles du framework des Collections en Java. Les classes proposées sont par exemple:
  - Sequence
  - Vector
  - Deque
  - Pair
  - Set
  - Stack
  - Queue
  - PriorityQueue

Utilisation de Ds:

- Les classes et interfaces de ds s'utilisent dans un espace de nommage **Ds**:

```
<?php
    $vector = new \Ds\Vector();
    $vector->push("un");
    $vector->push("deux");
    $vector->push("trois", "quatre");
    // ...[ , ] = unpacking
    $vector->push(...["cinq", "six"]);
    print_r($vector);
?>
```

Exemple d'utilisation de la classe *Set*

- **Question HTML dans un select multiple:**
- Prenons un petit formulaire en HTML qui propose un choix de couleurs dans un select:
- Quelles sont les couleurs RGB ?
  - <select name="coul[]" multiple size=5>
    - <option value="jaune">Jaune
    - <option value="rose">Rose
    - <option value="bleu">Bleu
    - <option value="noir">Bleu
    - <option value="blanc">Rouge
    - <option value="vert">Vert
  - </select>
-

- **Réponse PHP avec Collections**
  - <?php if (!empty($_GET['coul']))
  - { $couleursReponse = new \Ds\Set($_GET['coul']);
  - $couleursCorrectes = new \Ds\Set(['rouge',‘green','bleu']);
  - // Calculons la différence des 2 ensembles et voyons si elle est vide
  - if (($couleursReponse->diff($couleursCorrectes))->isEmpty())
  - echo "Bravo les couleurs RGB sont bien Rougee green et Blanc !";
  - else
  - echo "Mauvaise réponse : les couleurs RGB sont: Rose Noir et Blanc !"; }
- Sans Collections, on utilise les tableaux …
- Sans **Ds**, nous aurions été obligés de nous contenter de tableaux PHP et d’utiliser par exemple la méthode **array_diff**: Voir : array_diff php

Les variables d'environnements

- PHP dispose d'une série de variables déjà créées et qui stockent des données relatives à l'environnement d'exécution du code. On les appelle **variables d'environnement**.
- Les variables d'environnement sont très utiles en programmation (notamment en PHP). Il permettent d'avoir une idée claire sur l'environnement du programme qui peut être soit le client ou le serveur. Elles peuvent ainsi fournir des informations sur le type du serveur, son adresse IP, version des spécifications CGI utilisées, type du navigateur du visiteur, le nom de sa machine... etc
- Pour connaitre toutes les variables d'environnement disponibles on fait appel à la fonction **phpinfo()**.

Fonction phpinfo()

- La fonction phpinfo() permet d'avoir un récapitulatif sur la configuration de PHP sur le serveur Web. Elle affiche un tableau qui contient de nombreuses informations telles que la version du langage, options de compilation, extensions, entêtes HTTP, informations sur le système. Cependant, cette fonction est connue pour pouvoir englober l'ensemble des variables d'environnement disponibles associées à leurs valeurs courantes.

  Pour avoir tous ces détails il suffit d'exécuter le code:

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

Manipulation des variables d'environnement

- Il existe deux manières d'appeler les variables d'environnement:
  - **Variable $_SERVER**: c'est d'une variable superglobale. Il s'agit en fait d'un tableau associatif dont les clés représentent les éléments d'environnement dont on souhaite connaitre la valeur. **$_SERVER** est également appelée variable serveur.
  - **fonction getenv()**: Cette fonction accepte comme paramètre une chaîne de caractères qui est la même que l'on passe à la variable $_SERVER en guise de clé. Cette chaîne représente l'élément dont on cherche à connaitre la valeur.

Liste des variables d'environnement

Dans ce paragraphe, on va essayer de lister les variables d'environnement les plus fréquents. Si vous voulez connaitre toutes les variables existantes il suffit d'exécuter la fonction **phpinfo()**.

SERVER_NAME getenv("SERVER_NAME") ou $_SERVER["SERVER_NAME"]
Retourne le nom du serveur qui exécute le script.

SERVER_ADDR getenv("SERVER_ADDR") ou $_SERVER["SERVER_ADDR"]
Retourne l'adresse IP du serveur qui exécute le script.

SERVER_ADMIN getenv("SERVER_ADMIN") ou
$_SERVER["SERVER_ADMIN"]
Retourne l'email de l'administrateur du serveur.

REQUEST_METHOD getenv("REQUEST_METHOD") ou
$_SERVER["REQUEST_METHOD"]
Retourne la méthode utilisé pour la requête HTTP.

DATE_GMT getenv("DATE_GMT") ou $_SERVER["DATE_GMT"]
Retourne la date du serveur au format GMT.

DATE_LOCAL getenv("DATE_LOCAL") ou $_SERVER["DATE_LOCAL"]
Retourne la date du serveur au format local.

DOCUMENT_ROOT getenv("DOCUMENT_ROOT") ou $_SERVER["DOCUMENT_ROOT"]
Retourne le chemin de la racine du serveur.

HTTP_HOST getenv("HTTP_HOST") ou $_SERVER["HTTP_HOST"]
Retourne le nom du domaine du serveur.

HTTP_USER_AGENT getenv("HTTP_USER_AGENT") ou $_SERVER["HTTP_USER_AGENT"]
Retourne des informations sur le navigateur et le système d'exploitation du client.

REMOTE_ADDR getenv("REMOTE_ADDR") ou $_SERVER["REMOTE_ADDR"]
Retourne l'adresse IP du client qui appelle le script CGI.

PHP_SELF getenv("PHP_SELF") ou $_SERVER["PHP_SELF"]
Retourne le nom de la page exécutée.

HTTP_REFERER getenv("HTTP_REFERER") ou $_SERVER["HTTP_REFERER"]
Retourne le nom de la page qui a envoyé le client vers la page courante.

Constantes PHP

- **Les constantes prédéfinie en PHP**

Les constantes en général servent à rendre un code plus claire et facile à comprendre. En plus des constantes personnalisées que l'on peut créer à l'aide de la fonction define(), PHP dispose de ses propres constantes prédéfinies.

- **Les constantes PHP** sont des constantes prédéfinies par le langage. Il en existe beaucoup mais j'ai opté pour vous énumérer les plus fréquentes:

PHP_OS: retourne des informations sur le système d'exploitation installé sur le serveur.

PHP_VERSION: retourne la version du langage PHP installée sur le serveur Web.

__FILE__: (deux soulignés avant et après) retourne le nom du fichier en cours (qui exécute le script PHP).

__LINE__: (deux soulignés avant et après) retourne numéro de la ligne qui exécute la constante.


PHP

POO

- A partir de PHP 5, il y a un tout nouveau model objet. La gestion des objets en PHP a été complètement réécrite, permettant de meilleurs performances ainsi que plus de fonctionnalités.
- Chaque définition de classe commence par le mot-clé **class** , suivi par le nom de la classe, qui peut être quelconque à condition que ce ne soit pas un mot réservé en PHP. Suivent une paire d'accolade contenant la définition des membres et des méthodes. Une pseudo-variable **$this** est disponible lorsqu'une méthode est appelée depuis un contexte objet.

POO

- Définition simple d'une classe :

```
<?php
class SimpleClass
{
   // déclaration d'un membre
   public $var = 'une valeur par défaut';

   // déclaration de la méthode
   public function displayVar() {
     echo $this->var;
   }
}
?>
```

POO

- **Le mot clé new :**
  - Pour créer une instance d'un objet, un nouvel objet doit être créé et assigné à une variable. Un objet doit toujours être assigné lors de la création d'un nouvel objet à moins qu'un l'objet ait un constructeur défini qui lance une exception en cas d'erreur. Création d'une instance :

```
<?php
   $instance = new SimpleClass();
?>
```

Exemples

```
<?php
// Nous créons une classe « Personnage ».
class Personnage
{
    private $_force;
    private $_localisation;
    private $_experience;
    private $_degats;
    // Nous déclarons une méthode dont le seul but est d'afficher du texte.
    public function parler()
       {
          echo 'Je suis un personnage !';
        }
 }
$perso = new Personnage;
$perso->parler();
```

Exemples

```
<?php
class Personnage
{
  private $_force;
  private $_experience;
  private $_degats;
}
$perso = new Personnage;
$perso->_experience = $perso->_experience + 1; // que ce passe t'il avec cette instruction.
```

Exemples

Changer ce code pour qu'il augmente la variable experience

```
<?php
class Personnage
{
    private $_force;
    private $_experience = 3;
    private $_degats;
    public function augmente()
    {
        $this->_experience = $this->_experience +1;
        echo $this->_experience;
    }
}
$perso = new Personnage;
$perso->augmente(); // que ce passe t'il maintenant.
?>
```

POO

- **Le mot clé extends**
  - Une classe peut hériter des méthodes et des membres d'une autre classe en utilisant le mot clé extends dans la déclaration. Il n'est pas possible d'étendre de multiples classes, une classe peut uniquement hériter d'une seule classe de base.
  - Les méthodes et membres hérités peuvent être surchargés, à moins que la classe parent ait défini une méthode comme final. Pour surcharger, il suffit de redéclarer la **méthode avec le même nom** que celui défini dans la classe parent. Il est possible d'accéder à une méthode ou un membre surchargé avec l'opérateur **parent ::**

POO

- Le mot clé extends

```php
<?php
    class SimpleClass
      {
       // déclaration d'un membre
       public $var = 'une valeur par défaut';
       // déclaration de la méthode
       public function displayVar()
          {
           echo $this->var;
          }
      }
```

```php
// extension de la classe
class ExtendClass extends SimpleClass
{
    // Redéfinition de la méthode parent
    function displayVar()
     {
        echo "Classe étendue\n";
        parent::displayVar();
     }
}
$extended = new ExtendClass();
$extended->displayVar();
?>
```

POO

- **Constructeurs** : void __construct (mixed args , ...)
  - Les classes qui possèdent une méthode constructeur appellent cette méthode à chaque création d'une nouvelle instance de l'objet.
  - Note : Les constructeurs parents ne sont pas appelés implicitement si la classe enfant définie un constructeur. Si vous voulez utiliser un constructeur parent, il sera nécessaire de faire appel à parent::__construct().
- **Exemple** :

```
<?php
class BaseClass {
   function __construct()
     {
      print "In BaseClass constructor\n";
       }
}
```

```
class SubClass extends BaseClass {
   function __construct() {
      parent::__construct();
      print "In SubClass constructor\n";
   }
}
$obj = new BaseClass();
$obj = new SubClass();
?>
```

POO

- Destructeurs : void __destruct ( )
  - PHP 5 introduit un concept de destructeur similaire aux autres langages orientés objet, comme le C++ . La méthode destructeur doit être appelée aussitôt que toutes les références à un objet particulier sont effacées ou lorsque l'objet est explicitement détruit. Exemple avec un Destructeur

```
<?php
class MyDestructableClass {
       function __construct() {
            print "In constructor\n";
           $this->name = "MyDestructableClass";
                                       }
         function __destruct() {
                        print "Destruction de " . $this->name . "\n";
                                    }
                                   }
$obj = new MyDestructableClass(); ?>
```

POO

- Visibilité
  - La visibilité d'une propriété ou d'une méthode peut être définie en préfixant la déclaration avec un mot-clé : **public** , **protected** ou **private**. Les éléments déclarés publics (public) peuvent être utilisés par n'importe quelle partie du programme. L'accès aux éléments protégés (protected) est limité aux classes et parents hérités (et à la classe qui a défini l'élément). L'accès aux éléments privés (private) est uniquement réservé à la classe qui les a définis.

POO (exemple de visiblité d'une variable)

```
<?php
    /**
     * Définition de MyClass
     */
   class MyClass
      {
          public $publicc = 'Public';
          protected $protectedd = 'Protected';
          private $privatee = 'Private';

          function printHello()
                    {
                    echo $this->privatee;
                    echo $this->protectedd;
                    echo $this->privatee;
                    }
      }
```

L'opérateur de résolution de portée (::) :

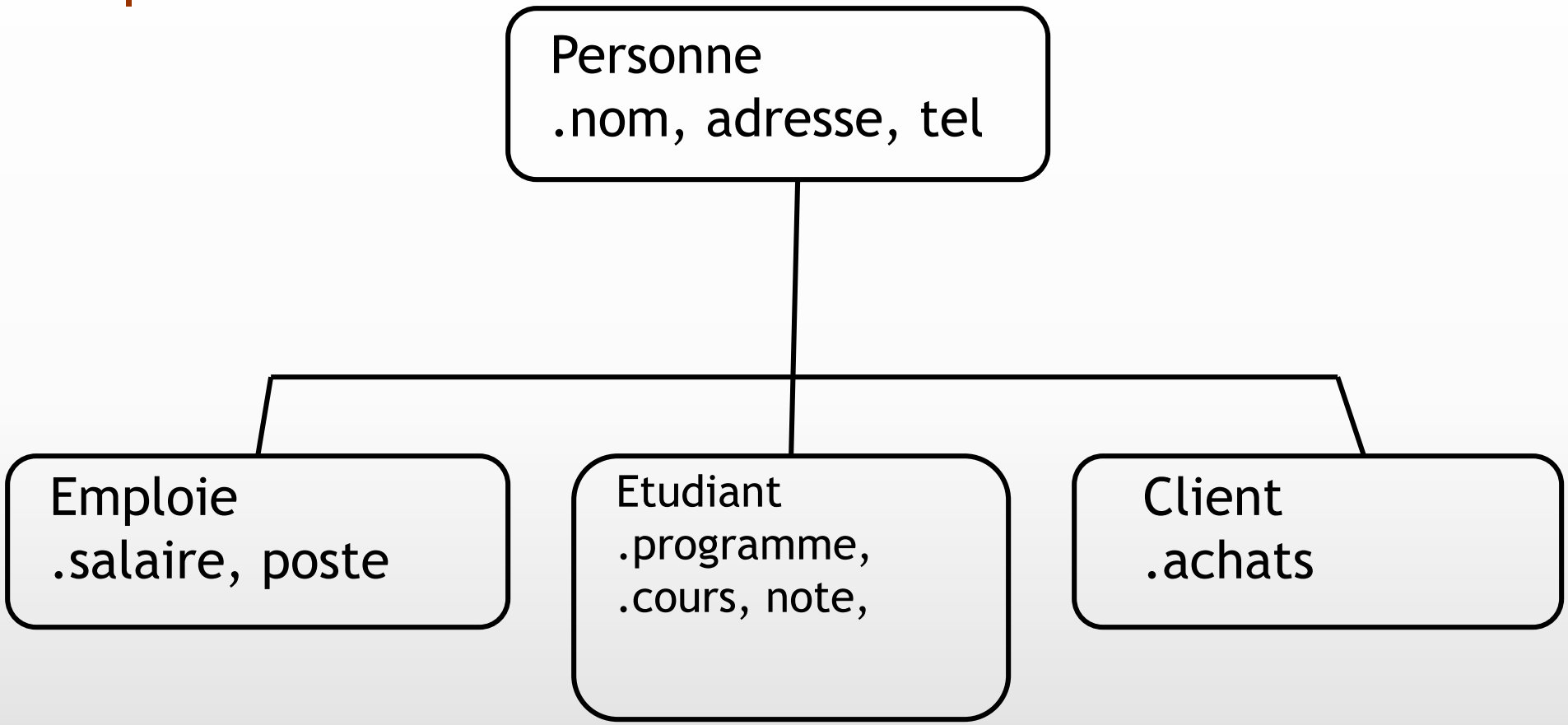
- Il fournit un moyen d'accéder aux **membres statiques ou constants** ainsi qu'aux éléments **redéfinis par la classe**.
- Lorsque vous référencez ces éléments en dehors de la définition de la classe, **utilisez le nom de la classe**.

```
<?php
        class MyClass {
                const CONST_VALUE = 'Une valeur constante';
        }
                echo MyClass::CONST_VALUE;
?>
```

L’héritage plus spécifiquement

- L’héritage consiste à réutiliser une classe de base (plus générale) pour en faire une version plus spécialisée.
- Par exemple, nous pourrions faire une classe Personne qui contient les propriétés nom et adresse pour ensuite dériver une nouvelle classe Employe à partir de Personne. Cette nouvelle classe permettrait d’ajouter la propriété salaire et les méthodes qui s’y rattache. Dans la même ordre d’idée nous pourrions dériver une autre classe Etudiant à partir de la classe Personne pour lui ajouter son dossier étudiant et les méthodes permettant de le manipuler.

L'héritage



Les classes dérivées héritent des propriétés nom, adresse et tel en plus de pouvoir utiliser les méthodes définies dans la classe de bas

La classe Personne

```
class Personne {
        private $nom;
        private $adresse;
        private $tel
        function __construct($n, $a, $t) {
                $this->nom = $n;
                $this->adresse = $a;
                $this->tel = $t;
                }
        function toHTML() {
                // …
                }
        }
        $toto = new Personne( "toto", "123 rue Laval", "514…");
        echo $toto->toHTML();
```

La classe dérivée Employe

```
class Employe extends Personne {
        private $salaire;
        private $poste;
        function __construct($n, $a, $t, $s, $p){
                parent::__construct($n, $a, $t);
                $this->salaire = $s;$this->poste = $p;}
        function toHTML() {// …}}
        $bob = new
        Employe("toto","123…","514…",45000,"prog");
        echo $bob->toHTML();
```

Terminologies

- Classe: type de données défini par le programmeur.
- Objet: instance de la classe. On défini la classe une seule fois et on l'utilise plusieurs fois pour créer des objets.
- Donnée membre: aussi appelée attribut ou propriété et représente une valeur faisant partie des données de la classe.
- Fonction membre: aussi appelée méthode et représente une fonction permettant d'agir sur les données de la classe.
- Classe parent: c'est la classe utilisée pour dériver une nouvelle classe. On l'appelle aussi classe de base.
- Classe enfant: c'est la nouvelle classe dérivée à partir d'une classe parent.

Encapsulation

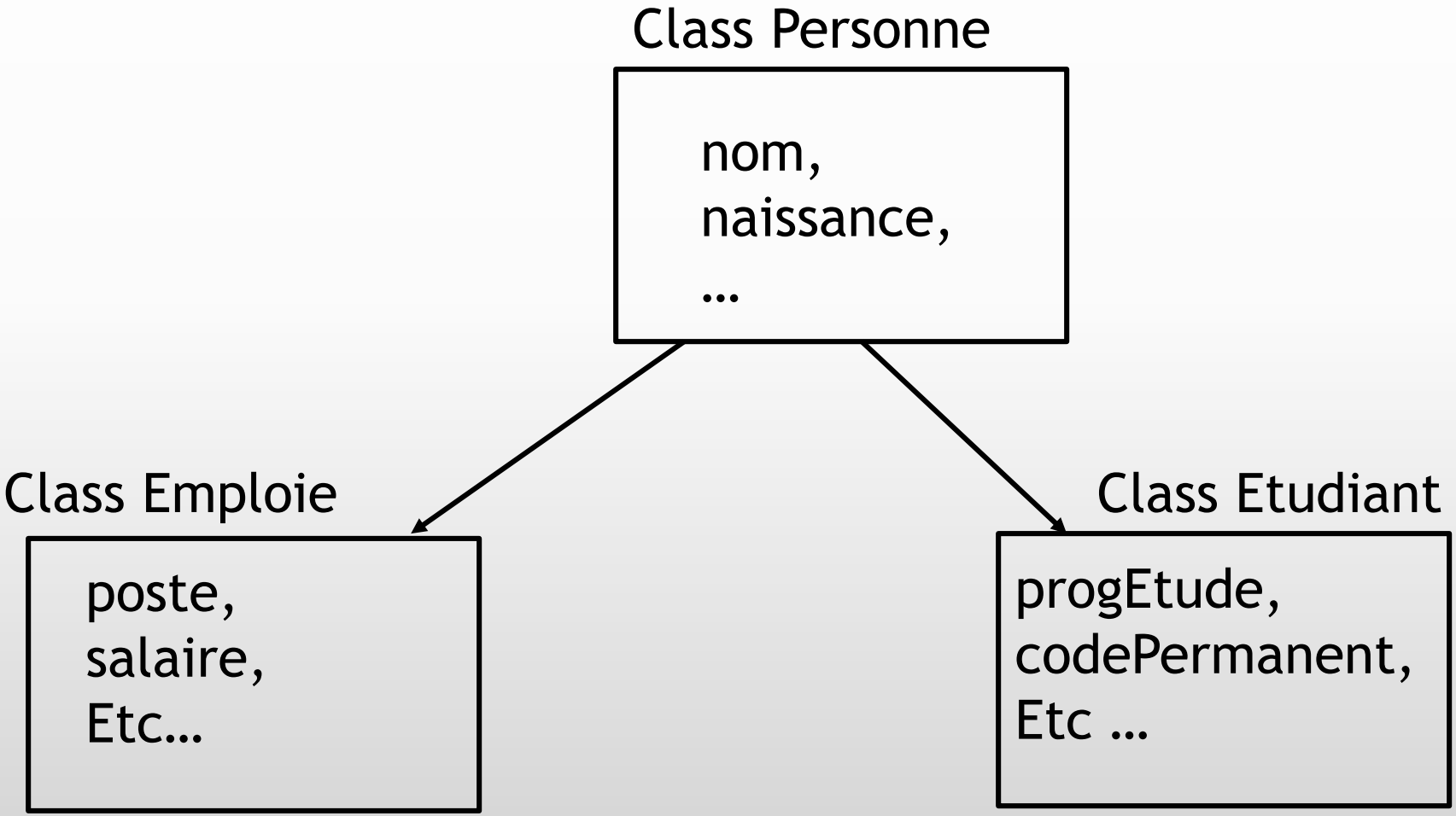
- L’encapsulation consiste à cacher les détails internes utiles au fonctionnement et à l’implémentation du type de données.
- Au niveau programmation on pourra décider de rendre disponible seulement certaines portions du type de données au monde extérieur et cacher le reste (les détails d’implémentation). De cette manière le concepteur qui décide de modifier la portion cachée peut le faire sans risque d’affecter les programmeurs l’utilisant car il est certain qu’ils ne voient pas cette portion.
- Il est également possible de forcer la modification des données en passant par un mutateur (fonction de modification) qui permettra de valider le changement avant qu’il soit effectuer.
- De la même manière il est possible de forcer la lecture en passant par un accesseur (fonction de lecture).

Héritage

- Le concept d'héritage est le plus important de la POO. Il permet la réutilisation de type de base défini par l'utilisateur tout en permettant de spécialiser le type.
- C'est la base des notions de réutilisation de composants logiciels.
- L'idée est de pouvoir définir (dériver) une nouvelle classe en se servant d'une classe existante (base).
- La classe dérivée hérite des membres de la classe de base tout en lui ajoutant de nouveaux.
- Il s'agit d'écrire de nouvelles classes plus spécifiques en se servant des définitions de base.
- Par exemple, nous pouvons dériver une nouvelle classe Employe en se servant de la classe de base Personne.
- Permet de définir la relation "est un". Par exemple, un employé est une personne ou encore un étudiant est une personne. Un cercle est une forme géo…


PHP

Héritage


Class Personne
nom,
naissance,
…
Class Emploie
poste,
salaire,
Etc…
Class Etudiant
progEtude,
codePermanent,
Etc …

Héritage

```
class Livre extends Produit {
    private $auteur;
    private $editeur;
    private $nb_pages;
    function _construct($auteur, $editeur, $nb_pages) {
        $this->auteur = $auteur;
        $this->editeur = $editeur;
        $this->nb_pages = $nb_pages; }
    function changer_nb_pages($nb){…}
    }
  $livre = new Livre(…);
  $livre->changer_nb_pages(150) ;
```

Constructeurs/destructeurs

- Un constructeur servira à initialiser le contenu d'un objet et même dans certains cas à allouer l'espace nécessaire aux données membres de l'objet.
- Le constructeur est appelé automatiquement lors de la création de l'objet, il est alors impossible de l'oublier.
- Un destructeur permet de libérer les ressources utilisées par un objet.

Constructeurs/destructeurs

- La classe est l'élément de base de la programmation par objets. Elle est le modèle à partir duquel des instances peuvent être créées.
- Une classe est un conteneur pour des propriétés (variables) et des méthodes (fonctions). On dit propriétés et méthodes membres

```
class NomDeLaClasse{
        var $propriete1;
        var $propriete2;
              function__construct() {
        //constructeur
                  }
        functionfonction() {
          // méthode   } }
```

Instanciation

- Afin d'instancier un objet, il faut utiliser le mot clé new $maVar = new NomDeLaClasse();
- Afin de faire référence à une propriété de l'objet, il faut utiliser le symbole ->

```
$maVar->prop = 23;
```

La classe Employe

```
class Employe {
        var $nom;
        var $salaire;
        function __construct($n, $s) {
                $this->nom = $n;
                $this->salaire = $s;
        }
        function toHTML() {
                return "<strong>Le nom:</strong><em>$this->nom</em>".
                "<strong>sal:</strong><em>$this->salaire</em>";
        }
    }
    $bob = new Employe( "Bob", 45000 );
    echo $bob->toHTML();
```

Méthodes statiques

- Certaines méthodes d'une classe peuvent être exécutées sans qu'on ait à créer une instance de celle-ci.
- Ces méthodes sont appelées statiques.
- Afin d'exécuter une méthode statique il faut la préfixer par le nom de sa classe, suivie de ::

```
LaClasse::laMethode();
```

Conventions

- Le nom d'une classe débute généralement par une majuscule. Ceux des attributs et méthodes par une minuscule.
- Ne manipulez pas les attributs d'une classe à l'extérieur de celle-ci, mais définissez des méthodes pour l'accès:
  - Accesseurs: pour lire un attribut getAtt()par exemple: echo $bob->getNom();
  - Mutateurs: pour modifier la valeur setAtt($var)par exemple: $bob->setNom("Larue");

Avantages des objets

- L'implémentation d'une classe peut être changée sans que les appels à celle-ci aient à être modifiés.
- La programmation par objets permet de créer du code très modulaire et réutilisable.
- Elle permet de penser en termes d'objets du domaine de l'application.

Self et parent

- Deux mots-clé spéciaux, **self** et **parent** , sont utilisés pour accéder aux membres ou aux méthodes depuis la définition de la classe.

```php
<?php
class MyClass {
    const CONST_VALUE = 'Une valeur constante';
}
echo MyClass::CONST_VALUE;
?>
```

```php
:: depuis la définition de la classe
<?php
class OtherClass extends MyClass
{
    public static $my_static = 'variable
                statique';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}
OtherClass::doubleColon();
?>
```

Statique

- Le fait de déclarer des membres ou des méthodes comme statiques vous permet d'**y accéder sans avoir besoin d'instancier la classe**. Un membre déclaré comme statique ne peut être accédé avec l'objet instancié d'une classe (bien qu'une méthode statique le peut).
- La déclaration static doit être faite après la déclaration de visibilité. Pour des raisons de compatibilité avec PHP 4, si aucune déclaration de visibilité n'est utilisée, alors le membre ou la méthode sera traité comme s'il avait été déclaré comme public .
- Comme les méthodes statiques sont appelables sans instance d'objet créée, la pseudo variable **$this n'est pas disponible dans la méthode déclarée en tant que statique**.

PHP

- **Statique :**
  - En fait, les appels de méthodes statiques sont résolus au moment de la compilation. Lorsque l'on utilise un nom de classe explicite, la méthode est déjà identifiée complètement et aucune notion d'héritage n'est appliquée. Si l'appel est effectuée par le mot clé self , alors self est traduit en la classe courante, qui est la classe appartenant au code. Ici aussi, aucune notion d'héritage n'est appliquée.
  - **On ne peut pas accéder à des propriétés statiques à travers l'objet en utilisant l'opérateur ->** .

Exemple avec un membre statique

```
<?php
class Foo
{
    public static $my_static = 'foo';
    public function staticValue() {
    return self::$my_static;
    }
}
class Bar extends Foo
{
    public function fooStatic() {
        return parent::$my_static;
     }
}

print Foo::$my_static . "\n";
$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n";
// propriété my_static non définie
// $foo::my_static n'est pas possible
print Bar::$my_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
?>
```

Constantes de classe

- Il est possible de définir des valeurs constantes à l'intérieur d'une classe, qui ne seront pas modifiables. Les constantes diffèrent des variables normales du fait qu'on n'utilise pas le symbole $ pour les déclarer ou les utiliser. Tout comme pour les membres statiques , on ne peut pas accéder aux valeurs constantes depuis une instance de l'objet (en utilisant $object::constant).

```
<?php
class MyClass
{
      const constant = 'valeur constante';
      function showConstant() {
      echo  self::constant . "\n";
                                        }
}
echo MyClass::constant . "\n";
$class = new MyClass();
$class->showConstant();
// echo $class::constant;  n'est pas autorisé
?>
```

Abstraction de classes

- PHP 5 introduit les classes et les méthodes abstraites. Il n'est pas autorisé de créer une instance d'une classe définie comme abstraite. Toutes les classes **contenant au moins une méthode abstraite doivent également être abstraites**. Pour définir une méthode abstraite, il faut simplement déclarer la signature de la méthode et ne fournir aucune implémentation.
- Lors de l'héritage depuis une classe abstraite, toutes les méthodes marquées comme abstraites dans la déclaration de la classe parent doivent être définies par l'enfant ; de plus, ces méthodes doivent être définies avec la même (ou plus faible) visibilité . Par exemple, si la méthode abstraite est définie comme protégée, l'implémentation de la fonction doit être définie en tant que protégée ou publique.

```
<?php
        abstract class AbstractClass
        …
```

- Interfaces
  - Les interfaces objet vous permettent de créer du code qui spécifie quelles méthodes et variables une classe peut implémenter, sans avoir à définir comment ces méthodes seront gérées.
  - Les interfaces sont définies en utilisant le mot clé interface , de la même façon qu'une classe standard mais sans aucun contenu de méthode.
  - Toutes les méthodes déclarées dans une interface doivent être publiques.
- implements
  - Pour implémenter une interface, l'opérateur implements est utilisé. Toutes les méthodes de l'interface doivent être implémentées dans une classe ; si ce n'est pas le cas, une erreur fatale sera émise. Les classes peuvent implémenter plus d'une interface en séparant chaque interface par une virgule.

- Interfaces

```php
<?php
// Declaration de l'interface 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}
```

```php
// Implémentation de l'interface
// Ceci va fonctionner
class Template implements iTemplate
{
    private $vars = array();
    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }
    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value,
$template);
        }
        return $template;
    }
}
```

Surcharge

- Les appels de méthodes et l'accès aux membres peuvent être surchargés via les méthodes **__call , __get et __set** . Ces méthodes ne seront déclenchées que si votre objet, hérité ou non, ne contient pas le membre ou la méthode auquel vous tentez d'accéder. Toutes les méthodes surchargées doivent être définies en tant que **public** .
- Depuis PHP 5.1.0, il est également possible de surcharger les fonctions isset et unset via, respectivement, les méthodes __isset et __unset.
- Mot clé ‘final’
  - PHP 5 introduit le mot-clé " final " qui empêche les classes filles de surcharger une méthode en en préfixant la définition par le mot-clé "final". Si la classe elle-même est définie comme finale, elle ne pourra pas être étendue.

final

The **methods** or **classes** can not be modified by a child class. This prevents class inheritance, method-overriding and/or redefinition of methods.
Only class definitions and/or methods inside a class can be defined as final. (not attributes)

Parcours d'objets

- PHP 5 fournit une façon de définir les objets de manière à ce qu'on puisse **parcourir une liste de membres** avec une structure foreach . Par défaut, toutes les propriétés visibles seront utilisées pour le parcours.

```
<?php
class MyClass{
public $a =9;
}
```

```
$class = new
MyClass();
foreach($class as
$key => $value)
```

```
{
    echo
"$key =>
$value \n";
}
?>
```

- Comme nous le montre l'affichage, l'itération foreach affiche toutes les variables visibles disponibles. Pour aller plus loin, vous pouvez implémenter l'interface interne de PHP 5 nommée **Iterator** . Ceci permet de déterminer comment l'objet doit être parcouru.

```
<?php
    class MyIterator implements Iterator
```

Auto-chargement de classes

- Jusqu'ici, nous avons défini les classes et nous les avons instancié pour nous servir des objets qui en sont les instances, et tout ceci dans la même page PHP. Bien que ça marche, ce n'est cependant pas une méthode de travail propre et organisée.
- Imaginons un instant qu'on aura besoin de plusieurs classes qui seront instanciées toutes dans le même document PHP. !!

La solution qui parait évidente consiste à déclarer chaque classe dans un document PHP à part. En effet, les développeurs PHP ont pour coutume de nommer la page qui contient la définition de la classe avec le même nom que celle-ci suivi de **.class** puis **.php**. Le **.class** sert à distinguer les pages de définition de classes des pages ordinaires.

- Imaginons par exemple qu'on veut créer une classe **String**. Le fichier qui la contiendra sera nommé: **String.class.php**.
- Le fichier de définition de la classe désormais créé, on peut alors l'inclure dans la page où on en aura besoin comme ceci:

```
<?php
  include("String.class.php");
?>
```

Auto chargement de classes (Autoload)

- Mieux encore, on peut créer une fonction qui se charge d'inclure la page souhaitée en lui passant, en argument, le nom de la classe comme ceci:

```
<?php
   function appel($classe){
      include($classe.".class.php");
   }
?>
```

- Pour inclure le fichier qui contient la classe String on fait alors:

```
<?php
   appel("String");
?>
```

- Cependant, un problème subsiste. En effet, si on veut inclure plusieurs classes, il faut appeler la fonction plusieurs fois. Cette opération est fastidieuse et en plus elle peut induire à l'erreur, car on pourrait bel et bien oublier d'appeler une page si ce n'était pas plusieurs.

Auto chargement de classes (Autoload)

- A partir de la version 5 de PHP, une fonctionnalité qui permet le chargement dynamique des classes est apparue permettant ainsi de faciliter au développeur l'inclusion des fichiers dont il aura besoin.
- La fonction **spl_autoload_register()** enregistre une fonction de notre choix dans la pile d'autoload. Cette pile est destinée à enregistrer des fonctions et les appeler, d'une manière implicite, quand on en aura besoin. De cette manière, si on fait appel à une classe (instanciation, héritage ou appel statique d'un membre) l'autoload se charge d'appeler la fonction qui permet d'inclure la classe souhaitée.
- Supposons que la fonction qui inclue les fichiers de classes est la **appel($classe)** vue précédemment:

```
<?php
  function appel($classe){
    include($classe.".class.php");
  }
?>
```

Pour enregistrer cette fonction dans la pile d'autoload on fait comme ceci:

```
<?php
  spl_autoload_register("appel");
?>
```

Autoload: Exemple

- Contenu de la page **String.class.php**:

```
<?php
  class String{
    private $str;
    private $length;
    public function __construct($chaine){
      $this->str=$chaine;
      $this->length=strlen($this->str);
    }
  }
?>
```

- Contenu de la page index.php (qui est placée, dans cet exemple, au même emplacement que String.class.php):

```
<?php
  function appel($classe){
    include($classe.".class.php");
  }
  spl_autoload_register("appel");
  $str=new String("Bonjour");
?>
```

Autoload: Exemple

- Au moment d'instancier la classe String, le fichier qui contient sa définition (String.class.php) est automatiquement inclus.
- Supposons maintenant qu'on exécute le code suivant:

```
<?php
   function appel($classe){
      include($classe.".class.php");
   }
   spl_autoload_register("appel");
   $str=new Date();
?>
```

- On obtient le résultat suivant:

Warning: include(Date.class.php) [function.include]: failed to open stream: No such file or directory in **index.php** on line **3**

- Vous avez remarqué que le compilateur est allé immédiatement chercher le fichier du nom **Date.class.php**. Cependant ce fichier n'existe pas.

Exercice

- Réalisez une classe Personne (class.Personne.php) qui permet de décrire un utilisateur à partir de son nom, prénom, login, mail, mot de passe, date de naissance, date d'inscrition, statut, etc.
- Après le constructeur, il faut coder les méthodes de :
  - mise à jour des informations updateData($info)
  - sauvegarde des informations saveData()
  - destruction du compte delDate()
  - Méthodes __get et __set pour les attributs privés


PHP

Formulaires et fonctions agissant sur les variables

- **Récupération des champs de formulaire**
- Supposons que nous disposons d'un formulaire déclaré comme ceci:

```
<form method="post" action="">
  <input type="text" name="prenom" /><br />
  <input type="submit" name="valider" value="Vérifier" />
</form>
```

- Bien entendu, c'est ce qu'il y a de plus simple en HTML. Il s'agit d'un formulaire qui contient deux champs, une zone de texte au nom de prenom et un bouton d'envoi du nom de valider. Le formulaire utilise la méthode POST et envoie ses valeurs à la page courante une fois le bouton d'envoi actionné.
- Sur la page qui est sensée traiter le formulaire (la page courante dans ce cas), on doit d'abord récupérer les valeurs postées par celui-ci. Pour ce faire, on recourt aux variables (tableaux) superglobales $_POST et $_GET.

Les Formulaires

- **Variables $_POST et $_GET**
  - La variables **$_POST** est en réalité un tableau associatif
  - La variables **$_POST** contient la valeur du champ de formulaire dont le nom est passé en tant que clé. Par exemple, pour récupérer la valeur que le client a saisi dans la zone de texte nommée **prenom**, on fait appel à la variable **$_POST["prenom"]**.

Les Formulaires

- La variable **$_POST** est appelée si le formulaire en question utilise la méthode **POST**. Si le formulaire utilise la méthode **GET** alors on appelle la variable **$_GET**. Tout comme le tableau **$_POST**, le tableau **$_GET** utilise comme clé, les noms des champs de formulaires.
- **$_POST** et **$_GET** sont des variables **superglobales**, c'est à dire qu'elles sont reconnues dans n'importe quel contexte (à l'intérieur des fonctions comme à l'extérieur, voir même à l'intérieur des méthodes d'une classe.

Les Formulaires

- **Fonctions agissant sur les variables**
  - Il existe une multitudes de fonctions qui agissent sur les variables en PHP. Mais les plus pratiques sont généralement:
    - **empty()**: permet de vérifier si la variable passée en paramètre est vide ou non. Si la variable est vide (elle ne contient aucune valeur) alors la fonction **empty()** retourne la valeur booléene **true**, et si la variable testée n'est pas vide (contient une valeur de type quelconque) alors elle retourne **false**.
    - **isset()**: permet de vérifier si la variable passée en paramètre existe ou non. Si la variable est déjà évoquée avant la fonction **isset()**, alors elle existe et cette dernière retourne **true**, sinon elle retourne **false**.

Les Formulaires

- **unset()**: permet de supprimer la variable passée en paramètre. Si après la fonction **unset()** on appelle la fonction **isset()** en leur passant la même variable, alors celle ci retournera **false**.
- **gettype()**: permet de retourner le type de la variable passée en paramtère
- **is_numeric()**: vérifie si la variable passée an paramètre ne contient qu'une suite de caractères numériques. Si oui elle retourne **true** sinon elle retourne **false**.
- **is_int()**: vérifie si la variable passée an paramètre est de type entier ou non. Si oui, elle retourne **true** sinon elle retourne **false**

- On trouve aussi les fonctions **is_float()**, **is_long()**, **is_double()**, **is_string()** et **is_bool()** qui vérifient chacune le type de variable associée de la même manière que **is_int()**.

Exemple d'application

- Pour faire simple, on suppose qu'on veut vérifier si le champ prénom du formulaire précédent n'est pas vide. Une fois le client validera le formulaire en cliquant sur le bouton d'envoi, si le prénom est vide, le message "Prénom invalide" sera affiché en rouge. Sinon (le prénom n'est pas laissé vide quelque soit le texte qu'il contient), alors le message "Prénom valide" sera affiché en vert. Quelque soit le message généré, il sera affiché en bas du formulaire.

Exemple d'application

```
<?php
  @$prenom=$_POST["prenom"];
  @$valider=$_POST["valider"];
  $message="";
  if(isset($valider)){
    if(empty($prenom))
      $message='<font color="#FF0000">Prénom invalide.</font>';
    else
      $message='<font color="#00FF00">Prénom valide.</font>';
  }
?>
<!DOCYPT html>
<html>
  <head>
    <meta charset="UTF-8" />
  </head>
  <body>
    <form method="post" action="">
      <input type="text" name="prenom" /><br />
      <input type="submit" name="valider" value="Vérifier" />
    </form>
    <?php
      echo $message;
    ?>
  </body>
</html>
```

Exemple d'application

- aucune des deux variables (tableaux) $_POST n'existe avant de poster le formulaire. Dans ce cas, au premier chargement de la page, les clés "prenom" et "valider" que contiennent les crochés ne seront pas reconnus par le compilateur PHP qui générera des erreurs de notification disant "Indefined index prenom" et "Indefined index valider". Ces erreurs n'interrompent pas l'exécution du script, mais la présence de ces deux messages d'erreur dans la page ne sera pas appréciée. Il existe donc deux solution,
  - soit interdire l'affichage des erreurs en manipulant le fameux fichier **php.ini** (chose fortement déconseillée lors de la phase développement)
  - ou bien masquer les messages d'erreur en utilisant le symbole arobas (**@**) avant l'instruction qui la cause. C'est cette deuxième méthode est la plus appropriée dans ce cas


PHP

PHP et les bases de données

- Supporte de nombreuses bases de données
  - DB2, Dbase, Oracle, SqlServer, ODBC, PostgreSQL, Unix DBM…
- Interface spécifique à chaque base
- Couche d'abstraction de bases de données
  - Exemple : PEAR:MDB2
- Principe d'interface similaire
  - Exemple : MySQL

Plusieurs moyens de se connecter à une DB MySQL

- L’extension mysql_ : ce sont des fonctions qui permettent d'accéder à une base de données MySQL et donc de communiquer avec MySQL. Toutefois, ces fonctions sont vieilles et on recommande de ne plus les utiliser aujourd'hui.
- L’extension mysqli_: ce sont des fonctions améliorées d'accès à MySQL. Elles proposent plus de fonctionnalités et sont plus à jour.
- L'extension PDO : c'est un outil complet qui permet d'accéder à n'importe quel type de base de données. On peut donc l'utiliser pour se connecter aussi bien à MySQL que PostgreSQL ou Oracle.

MySQL

- Ouvrir une connexion au serveur :

  mysql_connect
- Sélectionner une base de données de travail :

  mysql_select_db
- Effectuer une requête :

  mysql_query
- Lire le résultat d'une requête :

  mysql_result, mysql_fetch_array, ...
- Fermer la connexion :

  mysql_close

phpMyAdmin

- phpMyAdmin can manage a whole MySQL server as well as a single database over the World Wide Web.
- Official Site: http://www.phpmyadmin.net/
- Documentation: http://www.phpmyadmin.net/documentation/
- Features
  - Browser-based, Supporting PHP5.3+, MySQL 5.0+, Open Source
- There are five authentication modes offered:
  - http:
  - cookie
  - signon
  - config(the less secure one, not recommanded).
  - Swekey authentication mode:


```php
<?php
// Connexion et sélection de la base
$link = mysqli_connect('mysql_host', 'mysql_user', 'mysql_password')
    or die('Impossible de se connecter : ' . mysql_error());
echo 'Connected successfully';
mysqli_select_db($link,'my_database') or die('Impossible de sélectionner la
   base de données');
// Exécution des requêtes SQL
$query = 'SELECT * FROM my_table';
$result = mysqli_query($query) or die('Échec de la requête : ' .
   mysqli_error());
// Affichage des résultats en HTML
echo "<table>\n";
while ($line = mysqli_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n"; // ...
?>
```

PDO : Interface d'accès aux BDD

- PDO (PHP Data Objects) : interface parue avec PHP 5 qui vise à utiliser des bases de données sans avoir à s'occuper du SGBD utilisé derrière. Ainsi, il est tout à fait possible de faire un code qui marchera avec MySQL mais aussi avec Oracle, ODBC, etc.
- Avantages de PDO :
  - interface pour SGBD : plus besoin de s'occuper de savoir quelle SGBD est derrière (en théorie) ;
  - orienté objet : les objets PDO et PDOStatement peuvent être étendus, il est donc tout à fait possible de personnaliser et remodeler une partie du comportement initial ;
  - exception : les objets de l'interface PDO utilisent des exceptions, il est donc tout à fait possible d'intégrer facilement un système de gestion des erreurs.

PDO : Interface d'accès aux BDD

- **Activation** :
  - Avec PHP 5, cette extension n'est pas activée par défaut. Afin de remédier au problème, ouvrez le 'php.ini' que vous utilisez pour PHP 5 puis rendez-vous dans la partie où vous avez la liste des extensions. Faites une recherche sur '; Windows Extensions', vous devriez y arriver directement. Bien : maintenant, vérifiez que vous avez la ligne suivante :
    - ;extension=php_pdo.dll
  - Si c'est le cas, décommentez la ligne afin d'avoir :
    - extension=php_pdo.dll
  - Sinon, rajoutez la ligne à la main.
  - Vous venez donc d'activer PDO mais cela n'est pas suffisant : maintenant, il vous faut activer le driver correspondant au type de BDD que vous souhaitez utiliser.
  - Pour cela rien de plus simple : si vous voulez activer MySQL, rajoutez pour cela la ligne suivante (ou activez-la si elle est présente) :
    - extension=php_pdo_mysql.dll
  - Bien, enregistrez, relancez le serveur et vous voilà prêts à travailler avec l'extension PDO qui utilise MySQL

Instance PDO : Interface d'accès aux BDD

- Création d'une connexion :

```
<?php
//Instanciation d'un objet PDO
$dbh=new PDO(DSN [, user [, pass [, options]]]);
?>
```

  - DSN : Data Source Name
    - nom_du_driver:syntaxe_spécifique_au_driver
    - Ex: mysql:host=localhost;dbname=ma_base
  - user : nom d'utilisateur, pass : mot de passe
  - options : tableau associatif
    - spécifiques au driver
    - Ex : array(PDO::ATTR_PERSISTENT => true)) ;
  - Fin de connexion : $dbh=null ; ou unset($dbh) ;

Gestion d'erreurs éventuelles lors de l'instanciation (Exception PDOException)

- Création d'une connexion avec gestion des erreurs :

```
<?php
try
{
        $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
 }
catch(PDOException $e)
{
        echo 'Erreur : '.$e->getMessage().'<br />';
        echo 'N° : '.$e->getCode();
        die() ;
}
?>
```

Gestion des erreurs de connexion

```
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
    foreach($dbh->query('SELECT * from personne') as $row)
   {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Erreur !: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Inclusion de la connexion à la BDD

- On imagine que si notre projet Web est composés de plusieurs pages, alors la plupart d'entre elles aura besoin de dialoguer avec la base de données. Le code précédent sera donc déclaré dans chacune d'elles. Il faut alors songer à mettre ce code dans un fichier à part qu'on inclura au besoin (à l'aide de **include()** ou **require()**...).

PDO : Interface d'accès aux BDD

- **Requêtes** :
  - Comme vous le savez, il est possible sur une BDD de récupérer des informations, mais aussi d'effectuer des changements (qui peuvent prendre la forme d'ajout, suppression ou modification).
  - Si vous souhaitez récupérer une information (SELECT), vous devrez alors utiliser 'query' ; si c'est pour en apporter des changements (INSERT, UPDATE, DELETE) à la BDD, vous devrez utiliser 'exec'.

PDO : Interface d'accès aux BDD

- **Requêtes** :
  - Utilisation de la méthode : exec // Exécute une requête SQL et retourne le nombre de lignes affectées
  - Supposons que nous souhaitons modifier le mot de passe des membres :

```
<?php
        $conn = new  PDO('mysql:host=localhost;dbname=db', $user, $pass) ;

        /* Effacement de toutes les lignes de la table PRODUIT */
        $count = $conn->exec("DELETE FROM produit");

        /* Retourne le nombre de lignes effacées */
        print("Retourne le nombre de lignes effacées :\n");
        print("Effacement de $count lignes.\n");
?>
```

PDO : Interface d'accès aux BDD

- **Requêtes** :
  - Utilisation de la méthode : query
  - Imaginons que nous souhaitons connaître tous les membres :

```
<?php
$connexion=new PDO("mysql:host=$PARAM_hote;dbname=$PARAM_nom_bd", $PARAM_user, $PARAM_pass); // connexion à la BDD
$resultats=$connexion->query("SELECT membre FROM membres ORDER BY membre ASC"); // on va chercher tous les membres de la table qu'on trie par ordre croissant
$resultats->setFetchMode(PDO::FETCH_OBJ); // on dit qu'on veut que le résultat soit récupérable sous forme d'objet
while( $ligne = $resultats->fetch() ) // on récupère la liste des membres
{
      echo 'Utilisateur : '.$ligne->membre.'<br />'; // on affiche les membres
}
$resultats->closeCursor(); // on ferme le curseur des résultats
?>
```

Préparation et exécution des requêtes (Requêtes préparées)

- L'un des points fort de PDO est les **requêtes préparées**. En effet, une requête préparée est plus rapide et, aussi, plus sûre (surtout contre les tentatives d'injections SQL).
- Pour préparer une requête, il est recommandé de ne pas y spécifier les valeurs, mais plutôt remplacer celles-ci par des marqueurs interrogatifs (**?**) ou des paramètres nommées.

Exemple

- CREATE TABLE 'utilisateurs' (
  ' id ' int(10) unsigned NOT NULL auto_increment,
  ' date ' timestamp NOT NULL default
  CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  ' nom ' varchar(40) NOT NULL,
  ' prenom ' varchar(40) NOT NULL,
  ' login ' varchar(40) NOT NULL,
  ' pass ' varchar(40) NOT NULL,
  PRIMARY KEY (`id`)
  );
- On va inserer l'enregistrement: Nom:Einstein, Albert, login:a.Einstein, mot de passe:2022

Prepare & execute

- La méthode **prepare()** de l'objet PDO permet de préparer une requête. Elle accepte comme paramètre une chaîne de caractères qui correspond à la requête souhaitée. Les marqueurs interrogatifs sont là pour désigner les valeurs à passer à la requête. Ces valeurs seront ensuite passées lors de l'exécution de la requête. $ins est l'objet correspondant à la requête préparée.
- La méthode **execute()** est appelée par l'objet $ins qui correspond à la requête préparée. Elle accepte comme paramètre une variable de type tableau qui contient les valeurs qui remplaceront, dans l'ordre, les marqueurs interrogatif. Première entrée pour le premier marqueur, deuxième entrée pour le deuxième marqueur et ainsi de suite.
- Si la requête préparée ne contient aucun marqueur interrogatif alors la méthode **execute()** est laissée vide.

Marqueurs interrogatifs

- Nous allons donc préparer la requête puis l'exécuter comme ceci:

```
<?php
   $ins = $pdo->prepare("insert into utilisateurs (nom,prenom,login,pass)
values(?,?,?,?)");
   $ins->execute(array(
      "Einstein",
      "Albert",
      "a.Einstein",
      md5("2021") // criptage du mot de passe
   ));
?>
```

Explication

- Dans la requête préparée, on déclare à la place des valeurs des séquences qui commencent par deux points (:nom ou :prenom par exemple). Lors de l'exécution, on passe en paramètre un tableau associatif dont les clés sont les paramètres nommés (en tant que chaînes de caractères). Les valeurs du tableau seront les valeurs que l'on veut passer à la requête.
- Cette technique ne nous oblige pas de déclarer les valeurs au sein du tableau passé lors de l'exécution dans l'ordre établi dans la requête. C'est utile quand la requête contient un nombre important de paramètres, ce qui rend le respect de l'ordre peu compliqué.
- De la même manière on peut passer des requêtes de modification, suppression, vidage de la table...

Paramètres nommés

- Donc le traitement précédent peut être fait à l'aide des paramètres nommés au lieux des marqueurs interrogatifs comme ceci:

```
<?php
   $ins = $pdo->prepare("insert into utilisateurs (nom,prenom,login,pass)
values(:nom,:prenom,:login,:pass)");
   $ins->execute(array(
      ":prenom"=>"Albert",
      ":pass"=>md5("2020"),
      ":nom"=>"Einstein",
      ":login"=>"a.einstein"
   ));
?>
```

- Suppression de l'enregistrement inséré:

```
<?php
  $ins = $pdo->prepare("delete from utilisateurs where id=? limit 1");
  $ins->execute(array(1));
?>
```

- **Requête de sélection**
  - Pour passer une requête de sélection il y a un petit traitement à ajouter à ce que nous avons vu précédemment. Pour mieux comprendre, nous allons alimenter notre table avec des enregistrements à l'aide de la requête SQL suivante (que vous pouvez aussi passer directement via l'outil PHPMyAdmin)

- INSERT INTO 'utilisateurs' ('id', 'date', 'nom', 'prenom', 'login', 'pass') VALUES
(1,'2016-01-22 20:55:59','Einstein','Albert','a.einstein',
'81dc9bdb52d04dc20036dbd8313ed055'),
(2,'2016-01-22 20:55:59','Bohr','Niels','n.bohr',
'674f3c2c1a8a6f90461e8a66fb5550ba'),
(3,'2016-01-22 20:56:42','Hawking','Stephen',
's.hawking','f1f485b532be392dd7964f94dbd0562a');
- Supposons maintenant que nous souhaitons afficher toutes ces lignes dans une pages Web. On va donc procéder ainsi:
  - Tout d'abord on va préparer puis exécuter la requête de sélection:


```
<?php
   $ins = $pdo->prepare("select * from utilisateurs order by id");
   $ins->setFetchMode(PDO::FETCH_ASSOC);
   $ins->execute();
?>
```

Maintenant, il faut récupérer les enregistrements dans une variable PHP. En fait, on va récupérer toute la table dans une seule variable. Alors, il est évident que la variable sera un tableau à deux dimensions. Une pour désigner la ligne et l'autre pour désigner la colonne.

Revenons à la nouvelle méthode **setFetchMode()**. Il s'agit d'une méthode facultative appelée par l'objet requête et elle accepte comme paramètre une constante de classe. On utilise souvent **PDO::FETCH_ASSOC** pour dire que l'on veut récupérer les résultats dans un tableau associatif. Dans ce cas, le tableau récupéré aura dans le premier crochet le numéro de la ligne (commençant de 0) et dans le deuxième crochet une clé qui n'est rien d'autre que la valeur de l'attribut de la table (id, date, nom...).

- Le code qui permet de verser la table dans la variable tableau est le suivant:

```
<?php
   $tab = $ins->fetchAll();
?>
```

La méthode fetchAll() appelée par l'objet requête retourne un tableau contenant toutes les lignes retournées par la requête.

Désormais que nous disposons de la variable $tab, on peut en faire ce qu'on veut. Nous avons choisi d'afficher toutes les lignes sur la page. Le code ressemblerait donc à ceci:

```
<?php
  for($i=0;$i<count($tab);$i++){
     echo implode(" | ",$tab[$i])."<br />";
  }
?>
```

- Pour exploiter le caractère associatif du tableau on fait alors comme ceci:

```
<?php
   for($i=0;$i<count($tab);$i++){
      echo $tab[$i]["id"]." | ".
      $tab[$i]["date"]." | ".
      $tab[$i]["nom"]." | ".
      $tab[$i]["prenom"]." | ".
      $tab[$i]["login"]." | ".
      $tab[$i]["pass"]."<br />";
   }
?>
```

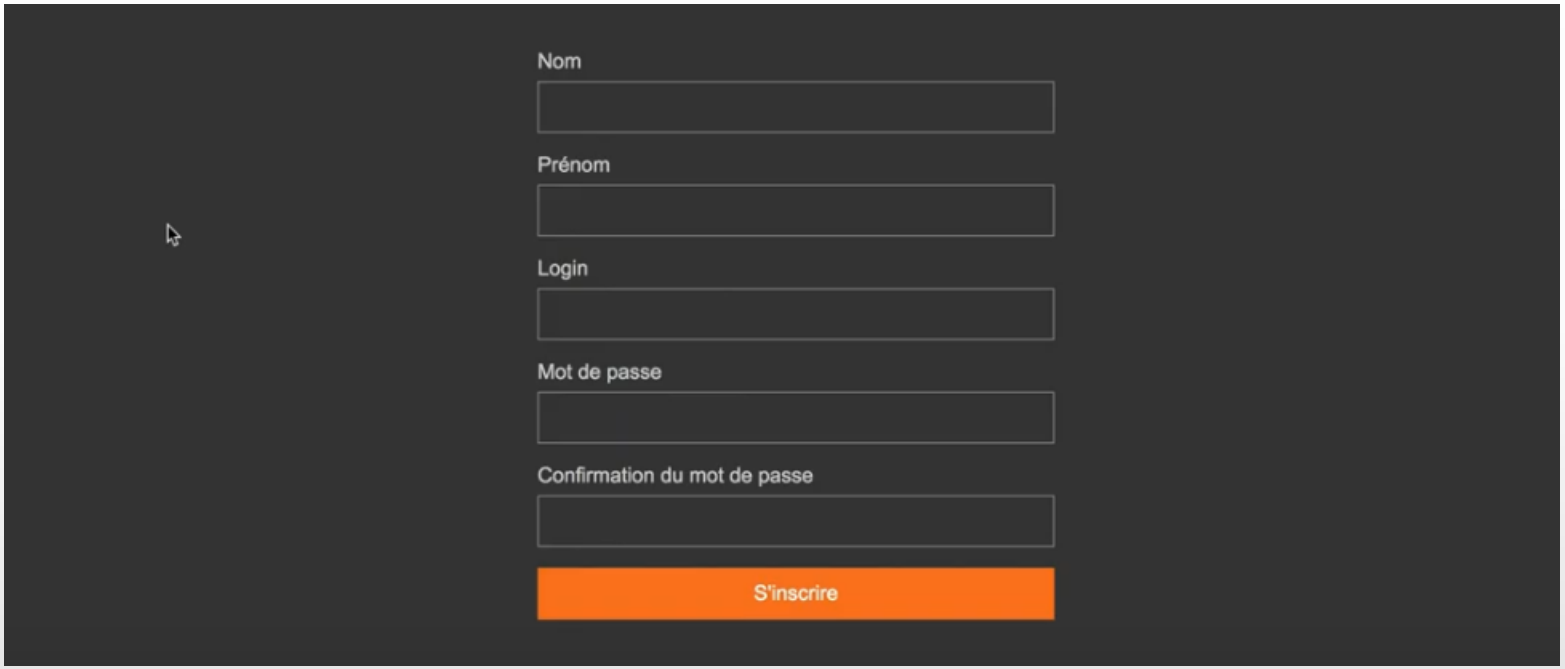
Fermeture d'une connexion

```php
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
// utiliser la connexion ici
$sth = $dbh->query('SELECT * FROM foo');

// et maintenant, fermez-la !
$sth = null;
$dbh = null;
?>
```


PHP

TD


Nom
Prénom
Login
Mot de passe
Confirmation du mot de passe
S'inscrire

cURL

- La librairie cURL permet de facilement communiquer avec de nombreux types de serveurs applicatifs en parlant le même langage que ceux-ci.
  - Ce langage est définit par ce qu'on appelle un protocole dont les plus connus sont sans aucun doute HTTP et FTP. L'extension cURL permet d'interagir en PHP avec tous ces protocoles que nous employons de manière quotidienne sans avoir à gérer la connexion ou encore sans se soucier de la manière dont il faut écrire la requête ou en recevoir la réponse.
  - Connection en HTTPS, FTP, SSL, Passage par un proxi
- Bibliothèque portable
- Communication entre serveur applicatifs
- Interaction transparente via les protocoles courants

cURL : protocoles

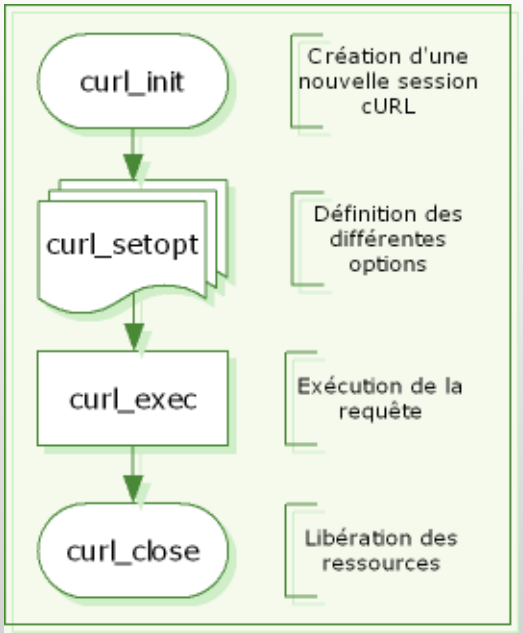
- Support de nombreux protocoles
  - HTTP: authentification, Get, Post, gestion des cookies et des sessions
  - FTP : authentification, liste, envoi et réception de fichiers
  - LDAP : authentification, lecture
  - Autres : Telnet, Gopher, Dict, File
- Chiffrage et certificat SSL
- Connexions via un proxy

cURL fonctionnement

```
<?php
/** Ce code récupère le flux ATOM de Google News et le
   retourne directement au navigateur **/
# Initialisation
$ch = curl_init("http://news.google.fr/"
   ."nwshp?hl=fr&tab=wn&output=atom");
# Exécution
curl_exec($ch);

# Fermeture de la session cURL
curl_close($ch);

# C'est tout !
?>
```



cURL

- cURL est utilisable dans d'autres langages
- cURL fait bien plus !
- Exemple :
  - récupération d'un flux ATOM dans un fichier temporaire et upload sur un serveur FTP distant


```
<?php
define('TMP_FILE', "tmp_atom.txt");
# Initialisation ... vide !
$ch = curl_init();
# Configuration de l'URL et du retour
curl_setopt($ch, CURLOPT_URL,
   "http://news.google.fr/nwshp?hl=fr&tab=wn&output=atom");
# Création du fichier de destination et ouverture rw
$fp = fopen(TMP_FILE, "w+");
# Lien : mettre le contenu ici
curl_setopt($ch, CURLOPT_FILE, $fp);
# Exécution
curl_exec($ch);
if(curl_errno($ch) != 0)  { echo "Erreur lors du téléchargement\n";
   die(); }
# Fermeture de la session cURL
curl_close($ch);
```


```
# Seek au début du fichier de destination
fseek($fp, 0);
# Et c'est reparti !
$ch_ftp = curl_init();
# L'emplacement FTP de destination
curl_setopt($ch_ftp, CURLOPT_URL,
   "ftp://ftp_login:password@ftp.exemple.fr/leFluxAtom");
# Indique à cURL qu'on uploade
curl_setopt($ch_ftp, CURLOPT_UPLOAD, 1);
# Transfert en ASCII
curl_setopt($ch_ftp, CURLOPT_TRANSFERTEXT, 1);
# Passage du pointeur
curl_setopt($ch_ftp, CURLOPT_INFILE, $fp);
# .. et la taille du fichier
curl_setopt($ch_ftp, CURLOPT_INFILESIZE, filesize(TMP_FILE));
# ET HOP !
curl_exec ($ch_ftp);
if(curl_errno($ch_ftp) != 0)  { echo "Erreur lors de l'upload\n";
   die(); }
curl_close ($ch_ftp);
echo "Terminé avec succès !\n"
?>
```

Concrètement: toHTML()

- On programme souvent une méthode toHTML() qui permet d'obtenir l'affichage HTML de l'objet.
- Il est généralement avantageux de laisser cette méthode retourner une chaîne de caractères (donc, on n'utilise pas echo dans toHTML()).
- L'utilisation la plus simple de toHTML() est donc:

```
echo $monObj->toHTML();
```

- Cette façon de faire simplifie l'utilisation de librairies de gestion de page qui deviennent de plus en plus la norme pour des projets d'envergure.

Concrètement: $_GETet $_POST

- Il est généralement avantageux de ne pas accéder directement $_GETet $_POST à l'intérieur d'objets si ceux-ci ne font pas partie de la couche de présentation («interface HTML»).
- Il vaut mieux ajouter des paramètres aux méthodes et passer $_GET ou $_POST comme paramètres aux fonctions

Concrètement: $_GET et $_POST

- Vous aurez alors des objets qui sont :
  - indépendants du protocole HTTP
  - qui peuvent être utilisés dans n'importe quel contexte (web, application indépendante, script, etc.)
  - pour lesquels des tests unitaires sont faciles à concevoir (parce qu'on n'a pas besoin de créer un contexte «web»).

Polymorphisme et héritage multiple

- Le polymorphisme est supporté partiellement en PHP 5. La redéfinition d'une méthode dans une classe enfant permettra de surcharger ce nom et la méthode correspondante sera appelée en fonction du type de l'objet utilisé.
- L'héritage multiple n'est pas supporté en PHP 5.


PHP

XML

- eXtensible Markup Language : langage de balisage extensible
- Assurer l'intéropabilité entre les SIs

```
<article>
   <title> Extensible Markup Language
</title>   <para>
        <acronym> XML </acronym>
        « langage de balisage
extensible »
   </para>
</article>
```

- Parsing d’un document XML
- Séparation présentation / contenu
- Analyse syntaxique
- Librairie *expat*

XML

- Deux types d'approche :
  - L'approche hiérarchique : DOM
  - L'approche événementielle : SAX

```
<debut> Bienvenue </debut>
```

Start Element : debut
Start CDATA section, value : Bienvenue
Close Element : debut

XML

- Créer l'analyseur : *xml_create_parser()*
- Association aux 7 gestionnaires :
  - *xml_set_element_handler()*
  - *xml_set_character_data_handler()*
  - *xml_set_external_entity_ref_handler()*
  - *xml_set_unparsed_entity_decl_handler()*
  - *xml_set_processing_instruction_handler()*
  - *xml_set_notation_decl_handler()*
  - *xml_set_default_handler()*

XML

- *xml_set_element_handler()* :
  - `function ouverture ($parser, $name, $attrs)`
  - `function fermeture ($parser, $name)`
  - `xml_set_element_handler($xml_parseur, "ouverture", "fermeture");`
- *xml_set_character_data_handler()* :
  - `function texte ($parser, $data_text)`
  - `xml_set_character_data_handler($xml_parseur, "texte");`
- *xml_set_default_handler()* :
  - `function defaut ()`
  - `xml_set_default_handler($xml_parseur,"defaut");`


```
<?php
/** Ce code récupère le flux ATOM de Google News et récupère
   les titres des articles **/
# Fonctions de retour (callback)
include('lib.sax.php');
$ch =
   curl_init("http://news.google.fr/nwshp?hl=fr&tab=wn&output
   =atom");
# Nous voulons récupérer le contenu dans une variable
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
# Exécution, contenu dans la variable $atomik
$atomik=curl_exec($ch);
if(curl_errno($ch) != 0)  { echo "Erreur lors de la
   récupération du flux\n"; die(); }
curl_close($ch);
```


```
# Création du parser
$sax=xml_parser_create();
/** Définition des fonctions de retour (callback) **/
# 1. Les tags
xml_set_element_handler($sax, 'openTag', 'closeTag');
# 2. Le contenu
xml_set_default_handler($sax, 'parseContent');

# Go go parsing
xml_parse($sax, $atomik, true); // premier et dernier morceau
echo xml_error_string(xml_get_error_code($sax));
?>
```


```php
<?php
/** Fonctions de callback **/
$printContent=false; // est ce qu'on doit afficher le contenu des balises ?
function openTag($parser, $name, $attribs)
{
    global $printContent;
    // traitement en fonction du tag ouvert
    switch(strtolower($name))
    {
        // ceci étant un exemple, on va écrire directement
        // et ne traiter qu'un échantillon de tags
        case 'title':
            echo '<h2>'; $printContent=true; break;
        case 'subtitle':
            // ... résultat : http://chipo.oxileo.net/~francois/sax.php
    }
}
```


```
function closeTag($parser, $name)
{
    global $printContent;
    switch(strtolower($name))
    {
        case 'title':
            echo '</h2>'; $printContent=false; break;
        case 'subtitle':
            // ...
    }
}

function parseContent($parser, $data)
{
    global $printContent;
    if($printContent) echo $data;
}
?>
```

Programmation objet

- TD (*partie 4*) :
  - Ecrire un formulaire qui récupère les informations de la base de données pour les injecter dans un fichier xml.
  - Générer le fichier xml personnes.xml

AJAX

- **AJAX** est un acronyme signifiant Asynchronous JavaScript and XML
- Permet d'appeler des données depuis un client Web sur un serveur en asynchrone
- AJAX nécessite une architecture client/serveur Web
- Composants conformes aux standards du W3C.

AJAX

- AJAX n'est pas une technologie mais l'utilisation conjointe d'un ensemble de technologies
  - **HTML** (ou XHTML) pour la structure sémantique des informations ;
  - **CSS** ou **XSL** pour la présentation des informations ;
  - **DOM** et **Javascript** pour afficher et interagir dynamiquement avec l'information présentée ;
  - l'objet **XMLHttpRequest** pour échanger et manipuler les données de manière asynchrone avec le serveur Web.
  - **XML** pour le transfert de données

AJAX

- Récupérer uniquement les données nécessaires via HTTP **XMLHttpRequest**
- Requêtes peuvent être « asynchrones »
- Applications plus réactives, la quantité de données échangées entre le navigateur et le serveur HTTP étant fortement réduite
- Chargement de la première page peut être pénalisé si l'application utilise une bibliothèque AJAX volumineuse

CONCLUSION ET REF

- Langage pour développer des pages Web dynamiques
- Très complet, nombreuses bibliothèques, facile …
- Possibilités de développements multi-serveurs, web services, pair à pair, etc.
- Références et compléments :
  - http://www.laltruiste.com/
  - http://php.developpez.com/cours/
  - http://www.expreg.com/ereg.php
  - http://www.manuelphp.com/
  - http://g-rossolini.developpez.com/tutoriels/php/cours/
  - http://www.php-mysql-tutorial.com/