PHP - Troisième année

Pierre BETTENS
pb@namok.be

ÉPSE - École de Promotion Sociale d'Enghien

22/11/2010

Organisation

- Cours
  - Exposé oral et manipulations
  - 160 périodes (133h 20m)
    - Réparties en deux modules de 80 périodes (66h 40m)
    - Ce cours est la deuxième partie
- Supports
  - Copie des transparents
  - Site web, http ://esi.namok.be
  - Références
    - Manuel PHP
    - *Learning PHP&MySQL*, Michele E.Davis & Jon A. Philips, Ed. O'Reilly
- Évaluation
  - Évaluation continue au cours des deux modules
  - Évaluation finale au terme de ce second module
    - 50% pour le cours
    - 25% cote artistique par Hélène Masse
    - 25% cote du jury de fin de 3^{e}

Organisation

- Objectifs
  - L'étudiant sera capable de structurer son code et son application.
  - Il sera conscient des éléments entrant en ligne de compte dans l'élaboration d'une application *web* utilisant PHP et ayant des données stockées dans une base de données
  - Il aura une approche qualité dans l'écriture de son code et dans le respect des conventions *web*
  - Il sera autonome et capable de rechercher la documentation seul

Menu

Préalables

Prérequis

Rappels

Préalables : : Prérequis

- Environnement de travail local (WAMP ou autre) et distant
- PHP
  - Types PHP, constantes et superglobals
  - Opérateurs, expressions, strutures de contrôle et structures répétitives
  - Fonctions, structuration du code
  - Cookie, session, authentification
- Utilisation de l'API PHP
- HTML élémentaire
  - Balises ...
  - Feuille de Style
  - Écriture de formulaire
- Persistence des données
  - Accès à une base de données
  - Conception de BD
  - Backup, portage

Préalables : : Rappels

- Convention d'écriture
  - Relire le document idoine [pdf]
- Organisation des fichiers
  - Fichiers propres à la gestion des données (BD)
  - Fichiers propres à l'aspect graphique
  - Autres documents (images, pdf, ...)
  - ...
  - *Regrouper les fichiers (bien nommés) dans des répertoires (bien nommés)*

OO, orienté objet

Introduction

Notions de classe

Notions d'instance

Syntaxe PHP

Méthodes magiques

Exceptions

Héritage

OO : : Introduction

- Lors du découpage du code, deux approches sont possibles
  - l'approche **procédurale** s'intéresse à ***ce qu'il faut faire***. Elle est orientée traitement
  - l'approche **orientée objet** s'intéresse aux données, ***que manipule-t-on ?***. Elle est orientée traitement **et** données
- **Définition** : Un *objet* est un élément logiciel ayant un ***état*** et un ***comportement***
  - Son *état* est défini grâce à ses **attributs** (taille, âge, couleur, position, ...)
  - Son *comportement* se reflète par les messages auxquels il peut répondre au travers de **méthodes** (grandir, changer de couleur, déplacer, ...)

OO : : Notion de classe

- **Définition** : Une *classe* est la structure permettant de définir un objet, elle est commune à tous les objets.
  Une classe rassemble les définitions des attributs et des méthodes.
- Exemple <img width="40

OO : : Notion de classe

- **Rappel** : Les *attributs* caractérisent l'objet (taille, prix, nom, couleur, ...)
- Les attributs d'une classe peuvent être **privés** (*private*), protégés (*protected*) ou **publiques** (*public*)
  - *private* : ne sont visibles qu'au sein de la classe
  - *protected* : visibles dans la classe et dans les classes enfants (voir plus loin, l'héritage)
  - *public* : visibles et accessibles de partout
- Les attributs d'une classe sont généralement **privés**
  - Les modifier "n'importe comment" risque de mettre l'objet dans un état incohérent (on utilisera des méthodes pour modifier l'état d'un objet)

```
$dagobert->price = -5;
```

OO : : Notion de classe

- **Rappel** : Les *méthodes* permettent d'agir sur une instance d'un objet (grandir permettra de changer l'attribut taille)
- Les méthodes d'une classe peuvent être **privées** (*private*), protégées (*protected*) ou **publiques** (*public*)
  - *private* : ne sont visibles qu'au sein de la classe
  - *protected* : visibles dans la classe et dans les classes enfants (voir plus loin, l'héritage)
  - *public* : visibles et accessibles de partout
- Les méthodes d'une classe sont généralement **publiques**
  - Afin de pouvoir y faire appel à l'extérieur de la classe
  - Permettent de modifier l'instance

```
$dagobert->increase_price(5);
```

  - Assure la cohérence de l'objet

```
$dagobert->setPrice(-5);
```

... devrait être refusé dans le code de la méthode

OO : : Notion de classe

- **Définition** : Le fait de rendre les *attributs privés* et les *méthodes publiques* s'appelle **l'encapsulation**
- L'encapsulation impose d'écrire des méthodes permettant d'accéder ou de modifer les attributs
  - Les **accesseurs** (*getters*) permettent d'accéder aux attributs

```
function getPrice() {
   return $this->price;
}
```

  - Les **mutateurs** (*setters*) permettent de modifier les attributs

```
function setPrice($price) {
    if ( $price >0 ) {
        $this->price = $price;
    }
}
```

OO : : Notion d'instance

- **Définition** : Une *instance* d'une classe est un objet **particulier**. Toutes les instances d'une même classe ont les mêmes attributs mais avec des valeurs différentes.
- Exemple <img width="50

OO : : Syntaxe PHP

- Une classe est définie en PHP par le mot clé `class`
  - Par convention son nom commence par une majuscule et on définira une classe par fichier (qui portera le même nom que la classe)
- Entre accolades se trouvent les définitions/déclarations des membres
- Exemple

```
<?php
class Sandwich {
    private $name;
    private $price;

    public function increase_price() {
      // ...
   }
}
?>
```

OO : : Syntaxe PHP

- Pour créer un objet (une **instance** de la classe), il faut utiliser le mot clé **`new`**

```
$instance = new Sandwich();
```

- Ceci a pour effet de créer un objet (une instance) avec des attributs **non**-initialisés, ils valent `NULL`
- Il existe une méthode particulière permettant de créer et d'initialiser un objet : **le constructeur**
  - `function __construct()`

OO : : Syntaxe PHP

- J'ajoute un constructeur à ma classe
- Le mot clé `this` permet de faire référence à l'instance "en cours" et l'opérateur `->` permet d'accéder aux attributs d'un objet
- ```
  function __construct() {
      $this->name = 'noname';
      $this->price = 0;
  }
  ```

  ou encore

  ```
  function __construct($name='noname', $price=42) {
      $this->name = $name;
      $this->price = $price;
  }
  ```
  ```

OO : : Syntaxe PHP

- L'opérateur **->** permet d'accéder à un attribut ou une méthode d'un objet
- Au sein de la classe

```
$dagobert->name
```

- À partir d'un autre script PHP

```
$dagobert->name = 'Jambon-fromage'     // Sauf si l'attribut privé
$dagobert->setName('Jambon-fromage');
```

OO : : Syntaxe PHP

- À ce stade, on peut manipuler un nouveau "type" ; un *Sandwich*

```
$dagobert = new Sandwich('Le Dagobert', 1.80);
$dagobert->increase_price(.1);

echo 'Un sandwich ' . $dagobert->getName() .
   ' coûte ' . $dagobert->getPrice();
```

OO : : Exercice

- **Exercice**
  - Écrire une classe `Sandwich`
  - Attributs
    - `name`, son nom
    - `price`, son prix
  - Méthodes
    - Les accesseurs (*getters*)
    - Les mutateurs (*setters*)
  - Écrire une page web affichant un objet de type sandwich (instancié "en dur" dans la page)

OO : : Méthodes magiques

- Les fonctions commençant par __ sont *magiques* en PHP,
- Principales fonctions magiques
  - `__construct` le constructeur
  - `__toString` détermine la représentation textuelle de l'objet
    - ```
      public function __toString() {
          return "$this->name ($this->price)";
      }
      ```
  - `__clone` copie l'objet
    - ```
      $copy = clone $objet;
      ```
    - Quelle différence avec `$copy = $objet`
  - ...

OO : : Méthodes magiques

- Quel différence entre *clone* et une affectation ?
- Affectation
  - Copie de la **référence** de l'objet, les deux variables référence le même objet. Modifier l'un, modifie l'autre.
- *Clonage*
  - Clone permet de faire une copie (superficielle) de l'objet, les deux varaibles référencent deux objets **différents** mais ayant les mêmes valeurs d'attributs

OO : : Exercice (suite)

- **Exercice**
  - Classe `Sandwich`
    - Ajouter un construteur à la classe Sandwich. Ce constructeur aura deux paramètres ayant comme valeur par défaut ; 'Sandwich anonyme', '0'
  - Écrire une classe `Tarif`
  - Attributs
    - `sandwiches`, une liste (*array*) de sandwiches
  - Méthodes
    - Un constructeur qui lira la liste des sandwiches dans un fichier csv
    - Une méthode `display_html` qui affichera tout les sandwiches dans une liste à puces

OO : : Exercice (suite)

- **Exercice**
  - Ajouter la possibilité d'ajouter un sandwich à la liste

OO : : Exceptions

- Le *mécanisme d'exceptions* permet de gérer différemment les erreurs qui peuvent survenir dans un programme.
- **Actuellement**
  - Le programme (le *script*) plante lamentablement ou

```php
$a = 1;
$b = 0;
echo $a / $b          // Affiche un Warning, divide by zero
```

  - s'arrête brutalement

```php
$link = mysql_connect($host, $user, $password)
    or die('Impossible de se connecter : ' . mysql_error());
```

OO : : Exceptions

- **Définition** : Le *mécanisme d'exceptions* permet au langage de s'interrompre –lorsqu'il détecte une erreur– et de **lancer** (*throw*) une exception. Cette exception pourra être **attrappée** (*catch*) afin d'être traitée.
- Remarques
  - Soit le langage soit le programmeur décide de lancer une exception.
  - C'est de la responsabilité du programmeur d'attraper les exceptions éventuellement lancées.
  - Ce mécanisme permet d'éviter l'arrêt brutal du script et autorise la reprise du code lorsque l'exception a été traitée
  - Ce mécanisme s'utilise pour gérer des évènements "exceptionnels" (défaillance d'une BD, erreur lors de la lecture/écriture d'un fichier, ...) par pour gérer la gestion courante (mauvaise saisie utilisateur, ...)

OO : : Exceptions

- Lancer (*throw*) une exception
  - L'instruction **`throw`** et l'utilisation de la classe `Exception` permettent de lancer une exception
  - Exemple

```
function div($a, $b) {
    if ($b == 0) {
        throw new Exception("Arguments invalides engendrent " .
            "une division par zéro");
    }
    return $a/$b;
}

print 'Division: ' . div(5,2) . "\n";
print 'Division: ' . div(5,0) . "\n";
```

OO : : Exceptions

- Attraper (*catch*) une exception
  - L'instruction `try catch` permet d'attraper une exception

```
try {
 // some code here
} catch (Exception $e) {
 // ...
}
```

  - Exemple

```
try {
    print 'Division: ' . div(5,0) . "\n";
    print 'Ce code n\'est pas exécuté';
} catch (Exception $e) {
    print 'Capture d\'une exception ' . $e->getMessage();
}
print 'Le script continue ici ...';
```

OO : : Exceptions

- Remarques
  - Il existe d'autres exceptions que `Exception` ... *cfr plus tard*

OO : : Héritage

- **Définition** L'héritage permet à une classe de récupérer (hériter) des méthodes et attributs d'une autre classe.
  - Permet de définir une classe par rapport à une autre (récupérer ce qui y est déjà défini)
  - La classe héritant (*classe enfant*) sera plus spécifique que l'autre (*classe parent*)
- Mot clé : `extends`
- Exemple
  - Une classe `Point` ayant comme attributs ; *abscisse* et *ordonnée*
  - Une classe `PointColoré` hérite de la classe `Point`
    - Attribut supplémentaire ; une *couleur*
    - Les attributs *abscisse* et *ordonnée* (et les méthodes) sont hérités

OO : : Héritage : : Exemple

- Classe `Point`

```php
<?php
class Point {
    private $x;
    private $y;

    public function __construct($x, $y) {
        $this->x = $x;
        $this->y = $y;
    }

    public function display() {
        echo 'Point (' . $x . ',' . $y .')';
    }
}
```

OO : : Héritage : : Exemple (suite)

- Classe `PointColoré`
- hérite de la classe `Point`
-

```
<?php
class PointColoré extends Point {
    private $color;

    public function __construct($x, $y, $c) {
        parent::__construct($x,$y);
        $this->$color = $c;
    }
}
```

OO : : Héritage : : Override

- **L'*override*** (réécriture) est la possibilité de réécrire (redéfinir) une méthode de la classe enfant, déjà écrite dans la classe parent

OO : : Héritage : : Override : : Exemple

```php
<?php
class PointColoré extends Point {
    // ...

    public function display() {
        echo 'Point (' . $x . ',' . $y . ') ' . $color;
}
```

OO : : Héritage

- **Remarque** Seuls les concepts utiles dans la suite du cours ont été exposés

PDO : : PHP Data Object

Introduction

Définitions

Transactions

Instructions préparées

Gestion des exceptions

API

PDO : : PHP Data Object

- **Définition** PDO est une couche d'abstraction OO permettant la gestion de la persistance des données *aka* les RDBMS
- Remarques
  - Par défaut cette extension est installée ...
  - ... il faudra cependant l'activer
    - Voir dans le fichier `php.ini`, une ligne de la forme `extension=pdo.so` (linux) ou `extension=php_pdo.dll` (MS windows)

PDO : : PHP Data Object

- Création d'un objet PDO par appel du constructeur
  - Le constructeur est suceptible de lancer une exception
- Exemple

```
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test',
         $user, $pass);
} catch (PDOException $e) {
    print 'Erreur de connexion à la BD' . $e->getMessage();
}
```

- Remarque : Il est possible de demander que la connexion soit **persistante**
  - Elle ne sera pas fermée en fin de script et pourra être réutilisée par un autre script
  - ```
    $dbh = new PDO('mysql:host=localhost;dbname=test',
        $user,
        $pass,
        array(      PDO::ATTR_PERSISTENT => true ));
    ```
    ```

PDO : : PHP Data Object : : Transactions

- PDO gère la notion de **transactions**
- **Définition**
  Les transactions offrent 4 fonctionnalités : *Atomicité, Consistance, Isolation* et *Durabilité* (ACID) ce qui permet d'avoir la certitude que tous les changements que vous voulez apporter à la BD seront fait en une seule fois.
- PDO offre les trois méthodes suivantes afin de gérer les transactions
  - `PDO::beginTransaction()`, permet de signaler que l'on débute une transaction
  - `PDO::commit()`, soumet la transaction au SGBD pour exécution
  - `PDO::rollBack()` revient en arrière (jusqu'au *begin*) si l'on rencontre une erreur

PDO : : PHP Data Object : : Transactions

- L'exécution d'une requête aura la forme

```
try {
  $dbh = new PDO('odbc:SAMPLE', 'DBNAME',
      'USER', 'PASSWORD',
      array(PDO::ATTR_PERSISTENT => true));
  echo "Connecté\n";

  $dbh->beginTransaction();
  $dbh->exec("ORDRE SQL");
  $dbh->exec("ENCORE UN ORDRE SQL");
  $dbh->commit();
} catch (PDOException $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
```

PDO : : PHP Data Object : : *Prepared statements*

- **Définition** Une requête préparée (*prepared statement*) est une requête contenant des variables qui pourront prendre des valeurs différentes lors d'exécutions consécutives de la même requête
  - Lors de l'exécution d'une requête le SGBD doit analyser/compiler/exécuter la requête. S'il supporte les *prepared statements* il peut économiser la répétition de ce travail
  - Les paramètres ne doivent pas être entre " ... ce qui permet d'éviter les *sql injection*
- Préparation de la requête

```
$stmt = $dbh->prepare(
    "INSERT INTO REGISTRY (name, value)
    VALUES (:name, :value)");
```

- Association du paramètre à une variable

```
$stmt->bindParam(':name', $name);
```

PDO : : PHP Data Object : : *Prepared statements*

- Exemple

```
$stmt = $dbh->prepare(
    "INSERT INTO REGISTRY (name, value)
    VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();

// insertion d'une autre ligne avec des valeurs différentes
$name = 'two';
$value = 2;
$stmt->execute();
```

PDO : : PHP Data Object : : Gestion des erreurs

- Par défaut PDO ne lance pas une exception lorsqu'il détecte une erreur, **3** manières de faire coexistent
  - `PDO: :ERRMODE_SILENT`, mode par défaut, silencieux. Il faudra explicitement demander s'il y a eu une erreur
  - `PDO: :ERRMODE_WARNING`, idem que précédement avec émission d'un message d'erreur traditionnel
  - `PDO::ERRMODE_EXCEPTION` lance une exception de type `PDOException`
- Définition du mode "exception"

```
$dbh->setAttribute( PDO::ERRMODE_EXCEPTION );
```

PDO : : PHP Data Object : : API

- `PDO:exec()` exécute une requête et **retourne le nombre de lignes affectées**
  - Valable pour des requêtes de type `update`, `insert`, ...
  - Pour obtenir les résultats, préparer une requête `PDO::prepare()`
  - ... et exécuter la *prepared statement* obtenue via `execute`
- `PDO::getAvailableDrivers()` retourne la liste des drivers disponibles
- `PDO::prepare()` prepare une requête SQL et retourne une `PDOStatement`
- `PDOStatement->exectute()` permet l'exécution d'un requête préparée
- `PDOStatement->fetch()` permet de récupérer la ligne suivante d'un *result set*
- `PDOStatement->fetchAll()` permet de récupérer tous les résultats dans un tableau

PDO : : PHP Data Object : : API

- `PDO::prepare()` prépare une requête SQL et retourne une `PDOStatement`
- Exemple

```
$stmt = $dbh->prepare(
    "INSERT INTO REGISTRY (name, value)
    VALUES (:name, :value)");
```

PDO : : PHP Data Object : : API

- `PDOStatement->exectute()` permet l'exécution d'un requête préparée

```
$stmt->bindParam(':name',$name);
$stmt->bindParam(':value',$value);
$name="Juste Leblanc";
$value=42;
$stmt->execute();
```

```
$stmt->execute(
    array('name' => 'Juste Leblanc',
        'value' => 42));
```

PDO : : PHP Data Object : : API

- `PDOStatement->fetch()` permet de récupérer la ligne suivante d'un *result set* sous différentes formes
  - Tableau associatif, *key-value* `PDO::FETCH_ASSOC`
  - Tableau indicé, *index-value* `PDO::FETCH_NUM`
  - Double tableau sur base d'un indice et d'un nom de colonne `PDO::FETCH_BOTH`
  - Associée les résultats aux variables liées (*bound*) via `bindParam` `PDO::FETCH_BOUND`
  - Associe les valeurs à un objet (anonyme ou pas)

PDO : : PHP Data Object : : API

- `PDOStatement->fetchAll()` permet de récupérer tous les résultats dans un tableau

PDO : : Exercice

- **Exercice**
  - Modifier dans la classe `Tarif` la persistance des données en utilisant une BD via PDO.

Pear Auth

Introduction

Installation

Exemple

API

Pear Auth : : Objet d'authentification

- **Définition** Pear Auth est un objet permettant la gestion de l'authentification
- **Définition** PEAR (PHP Extension and Application Repository) propose des librairies et des extensions PHP
  - Pear fonctionne comme un gestionnaire de paquets
  - Il est nécessaire d'installer PEAR et d'installer les paquets/modules/extensions que l'on désire utiliser

Pear Auth : : Objet d'authentification

- Installation de PEAR
  - Récupérer un script PHP permettant l'installation
    - `http://pear.php.net/go-pear`
    - Sauver le script localement (sous le nom `go-pear.php` par exemple
  - Exécuter le script
    - `php go-pear.php`
    - Suivre les instructions
  - Modifier le fichier `php.ini` afin d'inclure Pear dans le `PATH`
- Installation du paquet Auth
  - `pear install Auth`
  - `pear install MDB2`

Pear Auth : : Objet d'authentification

- Exemple d'utilisation
-

```php
require_once "Auth.php";

function loginFunction($username = null,
    $status = null, &$auth = null)
{ //... }

$options = array(
        'dsn' => "mysql://login:password@localhost/pearauth",
        );
$auth = new Auth("MDB2", $options, "loginFunction");
$auth->start();
if ($auth->checkAuth()) {
    /* Zone authentifiée *
}
?>
```

Pear Auth : : Objet d'authentification

- Formulaire d'authentification

```
// ...

function loginFunction($username = null,
    $status = null, &$auth = null)
{
echo <<<EOS
<form method="post" action="intro.php">
   <input type="text" name="username">
    <input type="password" name="password">
    <input type="submit">
</form>
EOS;
}
?>
```

Pear Auth : : Objet d'authentification

- C'est l'objet *Auth* qui se charge de la connexion à la base de donnée
- Création d'une table contenant les couples *login/password* valides
  - ```
    CREATE TABLE auth (
    username VARCHAR(50) default '' NOT NULL,
    password VARCHAR(32) default '' NOT NULL,
    PRIMARY KEY (username),
    KEY (password)
    );
    ```
    ```

Pear Auth : : Objet d'authentification

- Exemple de déconnexion d'un utilisateur
- ```
$myauth->start();
if ($_GET['action'] == "logout"
    && $myauth->checkAuth()) {
    $myauth->logout();
    $myauth->start();
}
?>
```
```

Pear Auth : : Objet d'authentification

- Il existe différents *container* pour l'authentification
  - DBMS (MySQL, ...) `Auth_Container_MDB2`
  - Un tableau de couples login/password `Auth_Container_Array`
  - Un annuaire LDAP (OpenLDAP, AD, ...) `Auth_Container_LDAP`
  - ...

Pear Auth : : Objet d'authentification

- Tutoriel Pear/Auth

AJAX

Préalables

Principes et fonctionnement

Exemple

AJAX

- **Définition** AJAX se base sur diverses technologies pour permettre à une page WEB d'exécuter une requête auprès du serveur **sans** recharger complètement la page
- Pour une bonne utilisation des resources, il est mieux de solliciter plus les clients que le serveur
- AJAX se base sur diverses technologies préexistentes et sur l'objet Javascript `XMLHttpRequest`
  - HTML / CSS
  - Javascript
  - PHP ou autre langage de script
  - L'objet **`XMLHttpRequest`**

AJAX

- L'objet **`XMLHttpRequest`**
  - Fonctionne en programmation événementielle
    - Les actions de l'utilisateur génère des évènements qui sont envoyés au serveur.
    - L'objet est à l'écoute de la réponse du serveur et peut réagir quand une réponse survient.
    - Le fonctionnement est **asynchrone**
  - Dispose de deux méthodes ; `open` et `send`
  - A la capacité d'exécuter une action lorsque l'état change `onreadystatechanged` *readyState* peut avoir les valeurs
    - 0 = uninitialized
    - 1 = loading
    - 2 = loaded
    - 3 = interactive
    - **4 = complete**

- L'objet **`XMLHttpRequest`**
  - Attributs
    - `readyState` : le code d'état passe successivement de 0 à 4 qui signifie "prêt".
    - `status` : 200 si tout va bien, 4040 si la page n'est pas trouvée
    - `responseText` : contient les données chargées dans une chaîne de caractères.
    - responseXml contient les données chargées sous forme xml, les méthodes de DOM servent à les extraire.
    - onreadystatechange propriété activée par un évènement de changement d'état. On lui assigne une fonction.
  - Méthodes
    - open(mode, url, boolean) mode : type de requête, GET ou POST url : l'endroit ou trouver les données, un fichier avec son chemin sur le disque. boolean : true (asynchrone) / false (synchrone). en option on peut ajouter un login et un mot de passe. send("chaine") null pour une commande GET.

AJAX : : Exemple

- Exemple d'utilisation d'AJAX permettant de remplacer un texte par le texte en majuscule.
- Cet exemple provient de ajaxf1.com

AJAX : : Exemple

- La page appelle un script JavaScript, `doWork`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta http-equiv="Content-Type" content="text/html;
      charset=utf-8" />
   <title>Ajax - PHP example</title>
</head>
<body>
   <form name="testForm">
      Input text: <input type="text" onkeyup="doWork();"
         name="inputText" id="inputText" />
      Output text: <input type="text" name="outputText"
         id="outputText" />
   </form>
</body>
</html>
```

AJAX : : Exemple

- Le script a besoin de l'objet `XMLHTTPRequest`

```
// Get the HTTP Object
function getHTTPObject(){
    if (window.ActiveXObject) {
        return new ActiveXObject("Microsoft.XMLHTTP");
    } else if (window.XMLHttpRequest) {
        return new XMLHttpRequest();
    } else {
        alert("Your browser does not support AJAX.");
        return null;
    }
}
```

AJAX : : Exemple

- Le script fait la requête auprès du serveur, par l'intermédiaire de `XMLHTTPRequest` et spécifie la méthode à exécuter au retour

```
// Implement business logic
function doWork(){
   httpObject = getHTTPObject();
   if (httpObject != null) {
      httpObject.open("GET", "upperCase.php?inputText="
         +document.getElementById('inputText').value, true);
      httpObject.send(null);
      httpObject.onreadystatechange = setOutput;
   }
}
```

AJAX : : Exemple

- Récupération de la valeur fournie par le serveur et utilisation sur la page
  - ```
    // Change the value of the outputText field
    function setOutput(){
        if(httpObject.readyState == 4){
            document.getElementById('outputText').value
                = httpObject.responseText;
        }
    }
    ```
    ```

AJAX : : Exemple

- Et du côté du serveur, le script PHP `upperCase.php`
  - ```php
    <?php
       if (isset($_GET['inputText']))
          echo strtoupper($_GET['inputText']);
    ?>
    ```
    ```

jQuery

Préalables

Mise en place

Principes

jQuery : : Préalables

- **Définition** : jquery est une librairie JavaScript (soit un ensemble de méthodes javascript)
- Permet de gérer plus facilement le contenu html d'une page, les évènement, les animations et AJAX
- Source : http ://docs.jquery.com

jQuery : : Mise en place

- Pour utiliser jQuery, il est nécessaire d'obtenir une copie de la librairie jQuery (ou d'en utiliser une en ligne)
- Ajouter ce code dans le header de vos pages
  - La librairie

```
<script type="text/javascript" src="jquery.js"></script>
ou
<script type="text/javascript"
    src="http://ajax.googleapis.com/ajax/libs/jquery/
                        1.4.1/jquery.min.js">
</script>
```

  - Le code personnalisé

```
<script type="text/javascript" src="customjquery.js"></script>
```

jQuery : : Mise en place

- Compléter les requêtes jQuery (dans le fichier `customjquery.js` par exemple)
  - ```
    jQuery(function() {
        $(document).ready(function(){
            $('#myclass').mymethod(
               // ...
                });
        });

        $('#myotherclass').myothermethod({
              //...
        });
    });
    ```
    ```

jQuery : : Principes

- Associer un sélecteur CSS à une action jQuery
  - Prévoir un sélecteur dans la page HTML
    - Un lien Link
    - Une classe CSS
- Ajouter une action sur le sélecteur dans le code JavaScript
  - ```
    $("a").click(function() {
        alert("Hello world!");
    });
    ```
    - `$("a")` est un sélecteur jQuery. Ce sélecteur agit sur chaque élement <a> de la page. On pourra utiliser des classes CSS (en cascade)
    - `click()` est une métode de l'objet jQuery (que l'on a instancié). Cette méthode lie les élements à l'action définie ensuite (dans la fonction de callback)
- On choisit un nom de classe et/ou un élément et on demande à jQuery d'agir sur la classe et/ou sur tout les élements
    ```

AJAX avec jQuery

- Il existe des méthodes jQuery permettant de manipuler l'objet `XMLHttpRequest`
  - Voir http ://api.jquery.com/category/ajax/
- Il suffit de préciser l'information que l'on envoie au serveur et de définir une fonction 'callback' précisant l'action à exécuter comme résultat

AJAX avec jQuery

- load
  - `jQuery.load( url, [ data ], [complete(responseText, textStatus, XMLHttpRequest) ] )`
  - Par défaut cette méthode remplit l'élément considéré par la valeur retournée. Si l'on fournit une méthode callback, son code est exécuté après
  - ```
    $('#result').load('ajax/test.html');
    ```

    ```
    $('#result').load('ajax/test.html', function() {
      alert('Load was performed.');
    });
    ```
    ```

AJAX avec jQuery

- post
  - `jQuery.post( url, [ data ], [success(data, textStatus, XMLHttpRequest) ], [dataType] )`
  - ```
$("#example a").click(function(e) {
    // Remove normal click action
    e.preventDefault();

   $.post("scriptname.php", {var: "value"}, function(xml) {
      // Traiter le contenu
      $("#examplereply").html("...");
    });
});
```
```

Framework PHP

Préalables

CodeIgniter

Installation

Modèle MVC

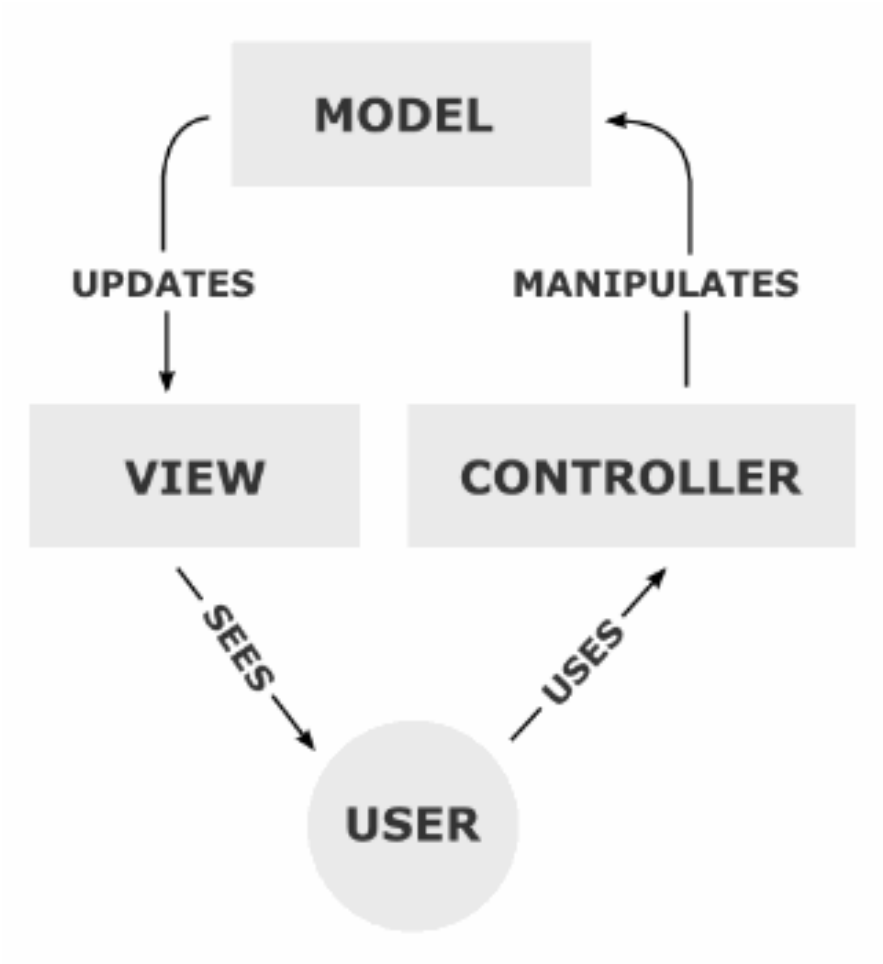
Divers

Framework PHP : : Préalables

- **Définition** Un *framework* est un ensemble de bibliothèques logicielles intégrées permettant un développement plus rapide et modulaire.
- Avantages et inconvénients
  - Offre une structure (souple ou rigide)
  - Demande un temps d'adaptation et d'apprentissage
  - Permet d'accélérer le développement

Framework PHP : : CodeIgniter : : Préalables

- CodeIgniter est un *framework* se basant sur le *design pattern* MVC
- **MVC** : modèle, vue, controleurs



- Le **modèle** contient la partie métier de l'application. (Généralement la gestion des données, et de la base de données)
- Une **vue** est la partie présentée à l'utilisateur. (Une vue est une page web ou un fragment de page (*header*, *footer*, flux RSS, ...))
- Les **controleurs** permettent à l'utilisateur d'agir sur le modèle
- Remarque : CodeIgniter permet de se passer d'un modèle si la complexité de l'application le permet

- Forme des URL
  - Une url se structure comme suit
    `http://example.com/news/article/my-article`
  - Correspond à *example.com/class/function/ID*
    - Où *class* est une "classe controleur",
    - *function* une méthode de la classe
    - et *ID* un paramètre passé à la méthode
- Un controleur au sens CodeIgniter est une classe qui correspondra à une page web
  - Si l'url `http://example.com/foo` est accessible alors il existe un controleur `foo.php`

- Se procurer l'archive et la dézipper
- *Uploader* le répertoire sur un serveur web
- Configurer
  - L'application (url, ...) via `application/config/config.php`
  - L'accès à la base de données via `application/config/database.php`

- Un controleur est une classe
  - se trouvant dans `applications/controllers/`
  - héritant de la classe `Controller`
  - contenant une méthode `index` (ou autre)
- Les controleurs peuvent être organisés en sous répertoires
- Le fichier `application/config/routes.php` contient le nom du controleur qui sera appelé par défaut
  `$route['default_controller']='anything';`
- **Remarque** : Une fonction commençant par un *underscore* ne sera pas visible et ne pourra donc pas être appelée via
  http ://example.com/class/_privatefunction

- Exemple *Hello World*
  - Ajouter un fichier `applications/controllers/helloworld.php`
    - ```
      <?php
      class HelloWorld extends Controller {

          function index() {
                  echo 'Hello World!';
           }

      }
      ?>
      ```
      ```

- Exemple *Hello World* (suite)
  - Aller à l'url `http://example.com/index.php/helloworld`
  - Ajouter une fonction `comment()` à la classe `HelloWorld`
    - aller à l'URL `http://example.com/index.php/helloworld/comment`
  - Ajouter une méthode `hello($name)` à la classe `HelloWorld`
    - aller à l'URL `http://example.com/index.php/helloworld/hello/Marlène`
  - Définir une URL par défaut, `$route['default_controller'] = 'HelloWorld';` dans le fichier application/config/routes.php

- **Définition** Une vue permet la mise en forme d'une page web. Elle est associée à un controleur.
  - Une vue représente une page web (ou un fragment)
  - Une vue n'est jamais appelée directement mais par le biais de son controleur
- Le chargement de la vue se fait donc dans le controleur
  - ```
    $this->load->view('myView');
    ```
  - Nécessite un fichier `myView.php` dans *applications/view*
- Les vues contiennent "le html"
    ```

- Plusieurs vues peuvent être chargées par le controleur, elles seront concaténées

```php
<?php

class Page extends Controller {

    function index()
    {
        $data['page_title'] = 'Your title';
        $this->load->view('header');
        $this->load->view('menu');
        $this->load->view('content', $data);
        $this->load->view('footer');
    }

}
?>
```

- Passage de paramètres du controleur vers la vue ... par le biais d'un tableau (ou d'un objet)
  - Dans le controleur
    - ```
      $data = array(
                      'title' => 'My Title',
                      'heading' => 'My Heading',
                      'message' => 'My Message'
                );

      $this->load->view('blogview', $data);
      ```
  - Dans une vue, on omet le nom du tableau, on utilise simplement les clés
    - ```
      <?php echo $heading;?>
      ```

- Exemple *Hello World*
  - Ajouter un fichier *applications/views/helloworld_view.php*
    - ```
      <html>
      <head>
          <title>Tutorial/essai de Code Igniter</title>
      </head>
      <body>
          <h1 >Hello world view</h1>
      </body>
      </html>
      ```
  - Modifier le fichier *applications/controllers/helloworld.php*
    - ```
      $this->load->view('helloworld_view');
      ```

- **Définition** Le modèle est la partie de l'application qui va traiter les données.
- Le modèle est une classe
  - se trouvant dans *applications/model*
  - héritant de la classe `Model`
    - ```
      class Model_name extends Model {

          function Model_name()
          {
              parent::Model();
          }
      }
      ```
      ```

- Un nom de classe doit commencer par une majuscule et le reste être en minuscule
- Le nom de fichier correspondant à la classe est le même mais tout en minuscules
- Le controleur utilise le modèle pour ce faire, il le charge :
  `$this->load->model('Model_name');`
- Une fois chargé, on peut accéder aux méthodes du modèle :
  `$this->Model_name->function();`

- Connection à la base de données
  - Le modèle ne se connecte pas automatiquement à la BD
  - Trois manières de se connecter
    - Utililser les paramètres de conf par défaut et préciser que l'on veut une connection `$this->load->model('Model_name', '', TRUE);`
    - Passer explicitement les paramètres de connection

```
$config['hostname'] = "localhost";
$config['username'] = "myusername";
$config['password'] = "mypassword";
$config['database'] = "mydatabase";
$config['dbdriver'] = "mysql";
$config['dbprefix'] = "";
$config['pconnect'] = FALSE;
$config['db_debug'] = TRUE;

$this->load->model('Model_name', '', $config);
```

    - Il est possible de se connecter automatiquement via `application/config/autoload.php`

- **Définition** Les aides (*helpers*) sont des classes qui facilitent la réalisation de certaines tâches
- Un *helper* n'est pas chargé par défaut, il doit l'être avant d'être utilisé.
  - ```
    $this->load->helper('name');
    ```
  - Il est possible de demander son chargement pour toutes les classes via `application/config/autoload.php`
- Exemples
  - *url helper*
  - *form helper*
  - *text helper*
  - *cookie helper*
  - *file helper*
    ```

- **Cache** Il est possible de mettre des pages en cache pour une durée déterminée. Pour ça, ajouter `$this->output->cache(n);` au controleur

Framework PHP : : CodeIgniter : : Liens

- Ressources
  - http ://codeigniter.fr
  - http ://codeigniter.com
  - http ://blog.angechierchia.com/ ?s=cms+code+igniter
  - http ://codeigniter.com/wiki/Category :Help : :Tutorials

Poins non abordés

- Certains aspects n'ont pas été abordés
  - Notion de *statique* et opérateur `::`
  - Classes abstraites
- Certains aspects pourraient être abordés prochainement ...
  - Connexion à LDAP
  - D'autres *frameworks* PHP

Crédits

- Slides générés grace à
  - Freeplane pour le *mind map* (carte mentale)
  - Script perso *latex2beamer* pour convertir le *mind map* en code LaTeX
  - LaTeXet consors pour en faire un joli PDF