

Solution n°23



Jean-Luc Collinet

Haute École Léonard de Vinci : Paul Lambin
Bloc 1 du Bac en Informatique

Semaine 3

Pour afficher le contenu d'une table

- Démonstration en pratique...
- Couche modèle : une méthode Db qui renvoie au contrôleur un tableau d'objets
- Couche contrôleur : un appel à la méthode Db et affectation d'une variable tableau
- Couche vue : parcours du tableau passé en variable du contrôleur à la vue

Protection de la faille XSS à l'affichage

- Utilisation d'htmlspecialchars dans le modèle, pour une vue
 - Déclaration de *getteur* "spécial vue" dans la classe

```
public function html_titre() {  
    return htmlspecialchars($this->_titre);  
}
```

- Dans la vue, utiliser le *getteur* "spécial vue"

```
<?php echo $tablivres[$i]->html_titre() ?>
```

Protection de la faille XSS à l'affichage

- Utilisation d'htmlspecialchars dans le contrôleur, pour une vue

- Dans le contrôleur

```
$html_motcle=htmlspecialchars($_POST['keyword']);
```

- Dans la vue

```
<input type="text" name="keyword"  
value="<?php echo $html_motcle ?>" />
```

Autres précisions

- Utilisons les fonctions **prepare**, **bindValue** et **execute** pour **tous** les types de requêtes SQL
 - Y compris le **SELECT**
- Faisons **appel** à **Db::getInstance()** **une seule fois** dans le contrôleur maître
 - Meilleure modularité des couches M et C

Cours de PHP



Jean-Luc Collinet

Haute École Léonard de Vinci : Paul Lambin
Bloc 1 du Bac en Informatique

A quoi servent les « *regex* » ?

- Valider de l'information donnée sous forme de chaîne de caractères

XOR (ne pas faire les 2 en même temps)

- Parser une chaîne de caractères pour y capturer de l'information
- Il est tout à fait déconseillé de parser de l'HTML avec les regex !

Expressions régulières

- Définir un ensemble de caractères
- Définir un ensemble de mots
- PCRE
 - Perl Compatible Regular Expression
- Adoptées dans plusieurs langages de programmation
- Syntaxe plus puissante et plus flexible que les expressions standards POSIX

Expressions de base

- `^http` : une chaîne qui commence par http
 - `be$` : une chaîne qui finit par be
 - `.` : un caractère, n'importe lequel
 - `[a-z]` : toutes les lettres minuscules de a à z
 - `[A-Z]` : toutes les lettres majuscules de A à Z
 - `[0-9]` : tous les chiffres de 0 à 9
-
- Nous allons toujours écrire un `^` et un `$`

Définir un nombre d'occurrence(s)

- **+** exprime **au moins une fois** ce qui précède
 - $^a+ \$$: au moins un a
- ***** exprime **0, 1 ou plusieurs fois** ce qui précède
 - $^b* \$$: pas de, 1 ou plusieurs b
- **?** exprime **0 ou 1 fois** ce qui précède
 - $^c? \$$: pas de ou 1 c
 - $^{\text{bon}}? \$$: bo ou bon (ce qui précède est 1 caractère)
 - $^{(\text{bon})} \$$: le mot vide ou bon (ce qui précède est entre parenthèses)

Préciser une cardinalité

- $\text{^Bla(bla)\{2\}\$}$

- Blabla

Non

- Blablabla

Oui : 2 fois le mot bla

- $\text{^Pf\{3,\}\$}$

- PfPfPf

Non : un seul car. pris

- Pfff

Oui : au moins 3 f

- Pfffffff

Oui

^ et caractères entre crochets

- `^[abc].*$`
 - Définit tous les mots qui commencent soit par un a, un b ou un c
- `^[^abc].*$`
 - Placé à l'intérieur des crochets, ^ signifie « ne contenant pas les caractères suivants... »
 - Définit tous les mots qui ne commencent pas par un a, un b ou un c

Le OU logique

- `^Mme|Melle$`
 - Représente-t-il l'ensemble {"Mme","Melle"} ?
 - Non : <https://regex101.com/>
 - commence par Mme ou se termine par Melle
- `^arti(san|ste)$`
 - Représente { "artisan" , "artiste" }

Exercices

- Définir tous les mots à 4 chiffres pouvant varier entre 0 et 9
 - $^[0-9]\{4\}\$$
- Définir tous les mots de 1 à 3 chiffres précédés éventuellement d'un signe négatif
 - $^-?[0-9]\{1,3\}\$$

Exercices

- Définir **toutes les adresses de courriel** correctement formatées
 - **Plusieurs réponses possibles** selon le degré de précision/complexité voulu
 - $^([a-zA-Z0-9]+(([\backslash.\backslash-\backslash_]?[a-zA-Z0-9]+)+)?)\@((([a-zA-Z0-9]+[\backslash.\backslash-\backslash_])+[a-zA-Z]{2,4}))\$$
- \backslash permet d'écrire le caractère spécial qui suit

Comment valider un mot ?

- Utilisez la fonction `preg_match`
- Extrait de code :

```
if (preg_match('/ expr.Rég. /', $chaine)) {  
    return true;  
} else { return false; }
```

 - `$chaine` est le mot à valider
 - Remplacez `expr.Rég.` par une expression régulière à satisfaire, écrite entre `/`

Parenthèses capturantes

```
# Transformation d'une date au format YYYY-MM-DD
```

```
# en une date au format DD/MM/YYYY
```

```
$date = "2020-2-21";
```

```
if (preg_match('/^([0-9]{4})-([0-9]{1,2})-  
                ([0-9]{1,2})$/', $date, $result))
```

```
{
```

```
    $dateout =
```

```
    "$result[3]/$result[2]/$result[1]";
```

```
}
```

L'option i : casse non sensitive

```
$fp = fopen("fichier.html", "r");
$content = fread ($fp, 2048);
# Capture des caractères entre les balises <title> et
#                                     </title>
if (preg_match('/<title>(.)+</title>/i',
$content,$res)) {
    $msg = "Le titre du fichier .html est : $res[1]";
}
fclose($fp);
```

D'autres moyens de valider...

- Fonction `filter_var`
- Quand la documentation est obscure
 - http://php.net/filter_var
- Clarifiez avec un site orienté tutoriel...
 - http://www.w3schools.com/php/filter_validate_email.asp

Bonne continuation à tous

- Le bœuf est lent,
mais la terre est patiente.



Sagesse paysanne