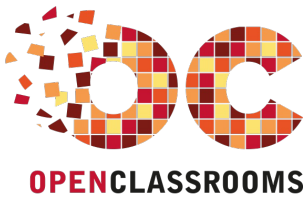
Utilisation de Twig, un moteur de templates !

Par robin850



www.openclassrooms.com

Dernière mise à jour le 8/07/2012

Sommaire

Utilisation de Twig, un moteur de templates !

Par  robin850

Mise à jour : 08/07/2012
Difficulté : Intermédiaire Durée d'étude : 3 heures

Bonjour à tous,

Aujourd'hui je vous propose de découvrir un moteur de templates du nom de Twig. Twig est le moteur de templates présent dans le célèbre framework Symfony. Ce cours portera sur son utilisation "seule" (c'est-à-dire sans Symfony) mais il faut savoir que la syntaxe que vous apprendrez à mettre dans vos templates sera valable pour ce cours et pour l'utilisation des templates avec Symfony (ou si le moteur est intégré dans un autre framework).

Twig est un moteur de templates PHP. Je vous conseille donc d'avoir de bonnes bases en PHP et si possible en orienté objet (bien que ça ne soit pas indispensable). Une petite note : la syntaxe du moteur est énormément inspirée de Jinja, le moteur de templates du framework web Python Django. Un cours est d'ailleurs disponible sur le site.

Je vous souhaite une agréable lecture !
Sommaire du tutoriel :

Qu'est ce qu'un moteur de templates ?

Twig n'est pas le seul moteur de templates qui existe, et ce n'est pas non plus le meilleur. C'est un moteur parmi tant d'autres. Au même titre que Twig, ce cours n'est pas le seul à traiter du sujet. Vous pouvez également lire ces tutoriels :

- Utilisation d'un moteur de templates : Talus' TPL
- Un moteur de template : Smarty

Présentation

Je vais ici vous expliquer le fonctionnement d'un "moteur de templates". Pour cela, je vous propose un exemple : la création d'un blog basique. L'organisation des fichiers que je vous propose ici est personnelle, c'est normal si pour vous la conception d'un blog est différente. Les grandes lignes sont là.

Pour créer votre blog, vous allez d'abord récupérer votre contenu stocké en base de données. Vous allez ensuite afficher via une boucle les différents articles. Dans certains cas, vous incluez aussi un en-tête et un pied de page.

Code : PHP - index.php

```
<?php include 'inc/header.php'; ?>

<?php

    $articles = $db->query('SELECT * FROM blog');

    while($article = $articles->fetch()) {

     ?>
        <div class="article">

            <div class="title"><?php echo $article['title']; ?
></div>
            <div class="content">
                <?php echo $article['content']; ?>
            </div>

        </div>
    }


<?php include 'inc/footer.php'; ?>
```

Je n'explique pas ce code. Il est relativement simple.

Le moteur de templates va vous obliger à utiliser deux fichiers (plus si vous utilisez le modèle MVC). Vous allez d'une part avoir un fichier PHP avec l'obtention du contenu en base de données, des variables, la gestion des sessions, etc. et un fichier template qui gérera l'affichage de vos pages. Normalement, plus aucun code HTML ne doit apparaître dans vos fichiers PHP.

Voici donc ce que **pourrait** donner notre exemple précédent avec un moteur de templates :

Code : PHP - index.php

```
<?php include 'inc/header.php'; ?>


    <?php

        $articlesQuery = $db->query('SELECT * FROM blog');
        $articles = $articlesQuery->fetchAll();

        $template = $twig->loadTemplate('blog.twig');
        echo $template->render(array(
      'articles' => $articles,
     ));
    ?>
<?php include 'inc/footer.php'; ?>
```

Code : HTML - blog.twig

```
{% for key, article in articles %}
 <div class="article">

            <div class="title">{{ article.title }}</div>
            <div class="content">
                {{ article.content }}
            </div>

        </div>
{% endfor %}
```

Je n'explique pas ce code pour le moment car vous allez apprendre cette syntaxe dans la suite du cours.

Pourquoi utiliser un moteur de templates ?

Depuis tout à l'heure je vous parle de Twig et je viens même de vous montrer les grandes lignes de son fonctionnement. Mais vous vous demandez peut-être pourquoi utiliser un moteur de templates ? Quel(s) avantage(s) peut-on avoir à l'utiliser par rapport à du PHP "classique" ?.

On peut y voir plusieurs avantages notables :

- Le code est plus clair : on sait où est le code HTML et on ne doit pas chercher après dans le code PHP
- Pour reprendre la phrase du dessus on peut noter le fait que vous savez quel fichier éditer quand vous savez d'où vient l'erreur. De plus pour reprendre l'exemple du blog, si un jour vous voulez modifier simplement l'affichage des billets, vous savez quelle partie du code éditer et vous n'avez pas à chercher.
- Si vous travaillez avec un graphiste, c'est un réel plus. Les graphistes connaissent généralement HTML et CSS et pas toujours PHP. Le graphiste s'y retrouve donc beaucoup plus facilement.

Je pourrais en citer d'autres mais je pense que ce sont les plus importants qui m'ont fait adopter un tel système pour des sites de petites et moyennes tailles.

Si je vous ai convaincu, je vous propose de lire la suite de ce cours pour voir comment mettre en place Twig.

Mise en place

Avant de mettre en place le moteur sur votre site, il vous faut le télécharger. Je vous propose donc de lire la partie qui vous concerne pour le faire. Si vous ne savez pas laquelle choisir, lisez la partie juste en dessous.

Via une archive

Pour télécharger Twig via une archive rien de plus simple, rendez vous sur le site et cliquez sur le lien "Install now". Choisissez ensuite votre archive (en tar ou en zip). Si vous ne savez pas trop, je vous conseille de prendre tar si vous êtes sur un système (de type) UNIX et zip sur un système Windows.

Vous devez maintenant décompresser l'archive et copier les fichiers dans un dossier nommé "twig" dans le dossier de votre site. Vous pouvez, si vous le souhaitez, l'appeler différemment ; en sachant que dans ce cours je pars du principe que vous avez appelé votre dossier contenant ces sources "twig".

Vous pouvez vous rendre à la partie "Mise en place"

Via git

Si vous voulez passer par Git, il vous suffit de vous placer dans le répertoire de votre site et de faire :

Code : Console

```
git clone http://github.com/fabpot/twig
```

Un dossier twig contenant les fichiers sources va être créé. Vous pouvez vous rendre à la partie "Mise en place"

Via Subversion

Pour cloner le dépôt SVN, placez-vous dans le répertoire de votre projet et tapez :

Code : Console

```
svn co http://svn.twig-project.org/trunk/ twig
```

Un dossier twig contenant les fichiers sources va être créé. Vous pouvez vous rendre à la partie "Mise en place"

Installation via PEAR

Pour les fans de PEAR, vous pouvez vous rendre sur la page dédiée à PEAR.

Note : à ma connaissance il n'existe pas de dépôt pour Mercurial ou pour CVS.

Mise en place

Si vous le souhaitez, vous pouvez supprimer tous les fichiers et dossiers sauf le dossier lib et son contenu. Je vous conseille de laisser le dossier tel quel et de ne pas changer les fichiers de place.

Twig est un moteur de templates réalisé avec le langage PHP en orienté objet. Il va donc falloir créer une instance des classes en question. Voici donc le code que vous devez mettre en début de chaque fichier (que je vous conseille de charger via include).

Avant tout, je vous demanderais de créer un dossier "templates" contenant les templates de votre site.

Code : PHP

```php
<?php
    include_once('twig/lib/Twig/Autoloader.php');
    Twig_Autoloader::register();

    $loader = new Twig_Loader_Filesystem('templates'); // Dossier
contenant les templates
    $twig = new Twig_Environment($loader, array(
      'cache' => false
    ));
```

Le code ici est assez simple. On inclut le fichier Autoloader.php. Ensuite, on indique le fichier où se trouvent nos templates. Pour finir, on demande à Twig d'aller chercher les templates dans le dossier indiqué précédemment et on lui indique quelques options pour plus de "souplesse" pendant le développement de notre projet.

D'ailleurs, ici je n'ai mis que le cache mais il y a d'autres options possibles :

- cache : prend en argument le dossier où vous stockez les templates en cache ou bien false pour ne pas s'en servir. Je me permet de vous donner un conseil : en production, le cache peut être une bonne chose mais en développement le mieux est de le mettre à false.
- charset : par défaut à utf-8, définit l'encodage de votre projet.
- autoescape : échappe automatiquement les variables. Le code HTML contenu dedans n'est donc pas interprété. Par défaut à `true`.

Pour plus d'options, je vous renvoie vers cette section de la documentation.

Sachez aussi que si vous le souhaitez, vous pouvez mettre plusieurs dossiers contenants les templates. En sachant que Twig va d'abord regarder dans le premier, puis le suivant et ainsi de suite. Je vous montre ici un code demandant à Twig d'aller chercher dans deux dossiers différents, mais vous pouvez en mettre plus :

Code : PHP

```php
<?php
$loader = new Twig_Loader_Filesystem(array('templates', 'views'));
```

Notre premier template

Je vous propose maintenant de voir la création de notre premier template. Avant toutes choses, créez un fichier appelé index.twig dans le dossier dans lequel vous stockez vos templates.

Ici j'ai mis "twig" comme extension mais en fait vous pouvez mettre ce que vous voulez. La plupart du temps, il s'agit de html ou tpl mais vous pouvez mettre txt, etc.

Ouvrez ce fichier avec votre éditeur favori et mettez-y le code suivant :

Code : HTML

```
Vous venez de créer votre premier template avec {{ moteur_name }} !
```

Nous reviendrons juste après sur cette portion de code. Ensuite, créez un fichier à la racine de votre site appelé index.php et mettez-y le code suivant (n'oubliez pas d'y inclure le fichier contenant le code qui instancie le moteur).

Code : PHP

```
<?php
    $template = $twig->loadTemplate('index.twig');
    echo $template->render(array(
 'moteur_name' => 'Twig'
    ));
?>
```

Vous pouvez aussi écrire ce qu'il y a au dessus de cette façon :

Code : PHP

```
<?php
    echo $twig->render('index.twig', array(
        'moteur_name' => 'Twig'
    ));
?>
```

Il n'y a pas énormément de différences mais cette portion de code est plus courte que la précédente.

Quand vous ouvrez le fichier, vous voyez "Vous venez de créer votre premier template avec Twig". En fait, dans le template le `{{ moteur_name }}` est le nom de la variable que vous avez passé à Twig lors de l'affichage du template. Il s'agit de cette portion de code.

Code : PHP

```
<?php
echo $template->render(array(
 'moteur_name' => 'Twig'
    ));
?>
```

Dans cet exemple, j'ai mis du texte entre guillemets, mais rien ne vous empêche de mettre des variables, des tableaux, etc.

Pour l'instant c'est plus que basique. Heureusement, Twig nous permet de faire bien plus de choses que ça. Je vous propose

donc de voir les possibilités que nous offre Twig dans la partie suivante.

Syntaxe de base

Cette partie est certainement la plus importante de ce cours. Vous allez apprendre à remplir vos templates avec la syntaxe proposée par le moteur. Vous pourrez donc faire des opérations complexes comme les conditions, les boucles, etc.

Affichage des variables et tableaux

Lors de l'affichage d'un template (vu un petit peu plus haut), vous passez en argument des variables, des tableaux, etc. Voyons un peu l'affichage côté template.

Code : HTML

```
Votre nom est : {{ name }}
```

Pour afficher une variable, il faut la placer entre accolades.

Note : si vous comptez utiliser Twig par la suite pour vos projets, je vous conseille de mettre un raccourci dans votre éditeur pour les doubles accolades et le tag {% ... %} que nous verrons après.

Supposons que vous voulez passer un tableau à votre template, il vous suffit de faire :

Code : HTML

```
{{ array['key'] }}
{{ array.key }} <!-- fait exactement la même chose qu'au dessus -->
```

Les filtres

Les filtres sont un concept que vous ne connaissez peut-être pas. En fait, un filtre agit comme une fonction. Il sont déjà définis mais vous pouvez créer les vôtres si vous le souhaitez. Pour appliquer un filtre à une variable, il faut séparer le nom de la variable et celui du filtre par un *pipe* (|). Je vous propose ici de voir quelques filtres utiles.

- upper : met la chaîne de caractères qui le concerne en majuscule ;
- lower : fait exactement l'inverse d'upper et va mettre toutes les lettres en minuscule ;
- title : met la première lettre de chaque mot de la chaîne de caractère en majuscule. Comme un titre de noblesse ;
- length : retourne le nombre de caractères dont la chaîne se compose ;
- escape & e : pour échapper du code HTML vous pouvez utiliser escape ou e. Tous deux font la même chose mais l'un est plus court que l'autre.

Il y a encore beaucoup d'autres filtres proposés par Twig, en voici la liste.

Les filtres sur les blocs

Si vous le souhaitez, plutôt que d'appliquer un filtre à chacune de vos variables, si vous comptez le faire sur beaucoup de variables ou une partie de votre site en particulier, je vous conseille de passer par cette méthode :

Code : HTML

```
{% filter upper %}
    je vais être écrit en majuscule
    {{ moi_aussi }}
{% endfilter %}
```

Vous pouvez aussi chaîner les filtres en en mettant plusieurs à la suite :

Code : HTML

```
{% filter upper|escape %}
    je vais être écrit en majuscule <em>et échappé</em>
{% endfilter %}
```

Concernant le filtre escape, un bloc spécial a été prévu à cet effet. Ainsi, si par exemple vous avez mis lors de l'initialisation de Twig autoespace à true, vous pouvez faire ceci :

Code : HTML

```
{% autoescape false %}
<strong>Je vais être affiché en gras</strong>
{% endautoescape %}
```

Les commentaires

Ici, c'est juste un petit plus. Ça n'est pas vraiment utile (c'est moins intuitif, dira-t-on) si vous n'avez pas l'habitude de les utiliser. C'est un plus pour ceux qui utilisent Django par exemple :

Code : HTML

```
{# Je suis un commentaire #}
```

Les conditions

Pour faire des conditions, rien de plus simple :

Code : HTML

```
{% if online > 1 %}
  Il y a {{ online }} membres en ligne
{% elseif online == 1 %}
    Il n'y a qu'un seul utilisateur en ligne (Vous !)
{% else %}
    Il n'y a aucun utilisateur en ligne. Cette phrase ne sera, en
principe, jamais lue sauf par toi. Ô grand développeur !.
{% endif %}
```

Les tests utiles (built-in tests)

Les tests utiles vous permettent de réaliser des conditions plus "poussées" et plus intuitives. Par exemple, avec, vous pouvez savoir si une variable est divisible par 10 ou si une variable est définie.

defined

Celui-ci vous permet de savoir si une variable est définie ou pas. Il renverra true si c'est le cas et false si ça ne l'est pas.

Code : HTML

```
{% if session.pseudo is defined %}
    Votre pseudo est {{ session.pseudo}} .
{% endif %}
```

divisibleby

Une test très intéressant est celui de savoir si votre variable est divisible par un certain nombre.

Code : HTML

```
{% if equipe.volley is divisibleby(6) %}
    L'équipe sera divisé en 2 équipes de 6.
{% endif %}
```

empty

Verifie si une variable est vide ou pas.

Code : HTML

```
{% if equipe.volley is empty %}
    Le match est annulé.
{% endif %}
```

Il y a encore beaucoup d'autres test-utiles mais je pense que vous avez compris le principe. Je vous renvoie donc vers la liste de ces tests.

La boucle for

Ici rien de bien compliqué. Je vais surtout vous donner des exemples car il n'y a pas grand-choses à dire sur cette boucle.

Code : HTML

```
{% for i in 0..50 %}
    Ceci est la ligne {{ i }}
{% enfor %}
```

En sachant que ce que je viens de vous montrer marche aussi avec des lettres :

Code : HTML

```
{% for lettre in 'a'..'z' %}
```

```
    La lettre {{ letter }} est lettre numéro {{ loop.index }} de
l'alphabet.

{% endfor %}
```

Si vous vous demandez d'où je sors le loop.index, il contient en fait le numéro de tour de boucle actuel. Vous pouvez voir d'autres variables de ce type ici (en scrollant un peu selon la taille de votre écran ; il s'agit du tableau).
Vous pouvez également parcourir vos tableaux avec :

Code : HTML

```
{% for joueur in club %}
    Le joueur {{ joueur.nom }} joue dans l'équipe {{ joueur.equipe
}}
{% endfor %}
```

Depuis la version 1.2 de Twig, il est possible de lancer une boucle selon une condition dans le même "bloc". Ce n'est pas très clair alors je vous propose un exemple.

Code : HTML

```
{% for joueur in club if equipe is not empty %}
    Le joueur {{ joueur.nom }} joue dans l'équipe {{ joueur.equipe
}}
{% endfor %}
```

Définir des variables

J'ai décidé de mettre cette partie en dernier pour une raison bien particulière. La logique d'un moteur de template veut qu'on sépare les variables et les fonctions php de l'affichage du contenu. Déclarer des variables dans un template casse la logique du système. Je vous le montre quand même car c'est quand même vous qui allez entretenir le code que vous écrivez par la suite (en principe) et vous faîtes comme vous le souhaitez.

Code : HTML

```
{% set var = "val" %}
```

Vous pouvez aussi définir des variables comme en Javascript. C'est à dire en mettant une seule fois "set" et en mettant à la suite les différentes variables séparées par des virgules :

Code : HTML

```
{% set mascotte, os = 'beastie', 'bsd' %}
```

Quelques ajouts pratiques

Voici quelques fonctions, pas indispensables, mais qui peuvent se révéler utiles.

Les includes

Vous pouvez aussi inclure des templates comme avec la fonction include de PHP. Pas besoin de vous écrire un roman :

Code : HTML

```
{% include 'header.twig' %}
```

Une chose que j'ai vraiment adorée c'est de pouvoir inclure des pages en leur restreignant l'accès à certaines variables.

Code : HTML

```
{% include 'fichier.twig' with {'var': 'val'} only %}
```

Ici, la page n'aura accès qu'à la variable `var`. Le `only` lui empêche d'accéder aux autres. Vous pouvez bien évidemment enlever le `only` pour que la page accède aux autres variables.

Une chose que j'ai aussi beaucoup aimée c'est la possibilité d'inclure des pages selon une condition. Par exemple :

Code : HTML

```
{% include online ? 'options.twig' : 'connexion.twig' %}
```

Un dernier point sur les include est le fait que vous pouvez inclure un template, lui passer des variables ou lui restreindre l'accès à certaines et indiquer à Twig que si le template est inexistant, aucune erreur ne sera renvoyée.

Code : HTML

```
{% include "sidebar.html" ignore missing %}
{% include "sidebar.html" ignore missing with {'foo': 'bar'} %}
{% include "sidebar.html" ignore missing only %}
```

Cet exemple est tiré de la documentation.

Les imports

On passe maintenant à une fonctionnalité que j'apprécie énormément : les imports.

Twig vous propose de créer un système équivalent aux helpers. Je vous copie le code de la documentation que je trouve tout simplement génial !
Supposons que cette page s'appelle forms.html :

Code : HTML

```
{% macro input(name, value, type, size) %}
    <input type="{{ type|default('text') }}" name="{{ name }}"
value="{{ value|e }}" size="{{ size|default(20) }}" />
{% endmacro %}

{% macro textarea(name, value, rows) %}
    <textarea name="{{ name }}" rows="{{ rows|default(10) }}"
cols="{{ cols|default(40) }}">{{ value|e }}</textarea>
{% endmacro %}
```

Et maintenant dans votre template, mettez le code suivant :

Code : HTML

```
{% import 'forms.html' as forms %}

<dl>
    <dt>Username</dt>
    <dd>{{ forms.input('username') }}</dd>
    <dt>Password</dt>
    <dd>{{ forms.input('password', none, 'password') }}</dd>
</dl>
<p>{{ forms.textarea('comment') }}</p>
```

Les formulaires sont un aspect récurrent et assez prise de tête quelques fois. Cette fonctionnalité permet un réel gain de temps !

Héritage

Une autre fonctionnalité qui m'a beaucoup plu est la possibilité d'héritage entre templates. Dans la plupart des cas, vous avez une charte définie pour votre site et à part le contenu, peu de choses changent dans la présentation de votre page. Avec Twig, vous pouvez définir un template avec votre header, votre footer (ce n'est qu'un exemple) et faire un héritage entre les deux templates (celui qui affiche le contenu et celui contenant le header et le footer) pour que vous n'ayez qu'à modifier un seul fichier si vous voulez changer la structure de votre site. Je vous prends encore un exemple de la documentation. En premier le template parent et en second le template qui hérite du premier (l'enfant).

Code : HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
  {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
  {% endblock %}
</head>
<body>
  <div id="content">{% block content %}{% endblock %}</div>
  <div id="footer">
    {% block footer %}
      &copy; Copyright 2009 by <a
href="http://domain.invalid/">you</a>.
    {% endblock %}
  </div>
</body>
</html>
```

Code : HTML

```
{% extends "base.html" %}

{% block title %}Index{% endblock %}
{% block head %}
  {{ parent() }}
  <style type="text/css">
    .important { color: #336699; }
  </style>
{% endblock %}
{% block content %}
  <h1>Index</h1>
  <p class="important">
```

```
        Welcome on my awesome homepage.
      </p>
    {% endblock %}
```

Dans cet exemple, le template parent s'appelle base.html. Un petit point dans le template enfant. Dans le block head, il y a un {{ parent() }}. Cette fonction signifie que le bloc n'est pas nettoyé et le contenu du bloc head sera celui présent dans le template parent et le template enfant. Pratique non ?

Fonctionnalités côté PHP

Pour l'instant je vous ai montré des fonctionnalités en majorité valables pour les templates. Je vous propose ici de voir certains points intéressants côté PHP.

Modifier les tags

Par défaut, Twig utilise la syntaxe des templates jinja. Pour afficher une variable il faut faire `{{ variable }}` et pour des instructions comme la boucle for, ou les conditions il faut faire `{% if confition %}`. Twig vous permet de modifier ces tags. Vous pouvez par exemple mettre les tags erb (moteur de template Ruby présent dans Ruby on Rails et Sinatra pour ne citer qu'eux) :

Code : PHP

```php
<?php
$syntaxe = new Twig_Lexer($twig, array(
    'tag_comment'  => array('#', '#'),
    'tag_block'    => array('<%', '%>'),
    'tag_variable' => array('<%=', '%>')
));
```

Ensuite, il faut indiquer au moteur que vous voulez modifier la syntaxe dans vos templates.

Code : PHP

```php
<?php $twig->setLexer($syntaxe); ?>
```

La variable `$twig` représente l'instance de la classe `Twig_Environnement()`. Si votre instance de la classe s'appelle différemment, changez en conséquence.

Étendre Twig

Les personnes utilisant des moteurs de templates sont nombreuses. Mais tout le monde ne l'utilise pas pour des projets de même taille. Twig peut donc être "étendu", c'est-à-dire qu'on peut lui rajouter des fonctionnalités très simplement grâce à des méthodes qui font tout le travail à notre place. Ça nous évite de toucher au code source pour ajouter un simple filtre par exemple.

Je vous propose ici de voir comment ajouter vos propres filtres et objets.

Ajouter un objet "global"

Un objet global, avec Twig, est un objet accessible depuis n'importe quel template. La documentation montre, par exemple, comment rajouter un objet "text" ayant une méthode lipsum. Je vous propose ici de voir comment intégrer cet objet.

Tout d'abord, il faut créer un fichier contenant une nouvelle classe :

Code : PHP

```php
<?php
class Text {

    public $lipsum_text = 'Lorem ipsum dolor sit amet';


    /**
* @param none
* @return lipsum_text
*/

    public function lipsum() {

        return $this->lipsum_text;

    }
}
```

Ensuite, indiquez à Twig que vous voulez ajouter votre objet dans les templates (il faut que le fichier contenant la classe soit inclus dans le fichier contenant cette instruction).

Code : PHP

```php
<?php $twig->addGlobal('text', new Text());
```

Le premier argument représente le nom de votre objet dans les templates et le second est une instance de la classe que vous avez précédemment créée. Et maintenant dans vos templates vous pouvez faire :

Code : HTML

```
Voici un texte : <br />
{{ text.lipsum() }}
```

Ajouter des filtres

Maintenant que vous avez compris le principe pour les objets, les filtres vont aller tout seuls. C'est la même marche à suivre que précédemment. Ici vous créez une fonction (pas forcément une méthode de classe) et vous demandez à Twig de l'ajouter au parseur. Ici je vais vous montrer comment ajouter le filtre lower (même s'il existe déjà, c'est juste pour l'exemple).

Code : PHP

```php
<?php $twig->addFilter('lower', new
Twig_Filter_Function('strtolower')); ?>
```

Ici, je n'ai pas eu à créer la fonction car elle est native mais vous pouvez mettre les vôtres sans aucun problème. Vous pouvez aussi appeler la méthode d'une classe mais **attention**, il faut que la méthode soit statique. Par exemple :

Code : PHP

```php
<?php $twig->addFilter('monfiltre', new
Twig_Filter_Function('MaClasse::MaMethode')); ?>
```

Ce cours s'achève là. Je vous invite à poster un commentaire si vous avez des suggestions, des points que vous ne trouvez pas clairs, etc. Pour ce qui est de l'aide, je vous renvoie vers le forum php.

Avant de vous quitter, je peux vous conseiller quelques liens :

- Symfony2 - Un tutoriel pour débuter avec le framework Symfony2 : un cours sur le célèbre framework Symfony dans sa version 2. Le framework utilise Twig.
- La documentation Twig : à mettre en favoris si vous comptez utiliser le moteur.
- Le blog : Vous pouvez voir ici les actualités sur Twig (dans 90% des cas, il s'agit de l'annonce d'une nouvelle version).
- La page de rapports de bug
- Dépôt Github : ce dépôt contient les sources du projet. Ça peut être intéressant de voir comment fonctionne le moteur.