# Le moteur de templates : Twig - Cours -

## Table des matières

# I. Le moteur de templates Twig.

## 1. *Qu'est-ce que Twig ?*

Twig est un moteur de templates développé par SensioLabs et plus particulièrement Fabien Potencier (qui a aussi développé le Framework PHP Symfony et Symfony2). Ainsi il est installé et utilisable par défaut dans chaque projet Symfony2.

Il peut cependant être installé et utilisé dans un simple projet PHP hors de Symfony.

Il est apparu pour la première fois sur la toile le 12 Octobre 2009.

Sa syntaxe est très inspirée de Jinja le moteur de templates du Framework Python Django. Il a été développé en PHP.

## 2. *C'est quoi ? A quoi ça sert ?*

Le rôle d'un moteur de templates est principalement de vous aider dans la lisibilité et la logique de votre projet en général et de son code en particulier. Également couplé d'une structure MVC, ce système donne d'excellentes performances.

Ce que fait précisément un moteur de templates, c'est rassembler le code de présentation (tout ce qui est HTML et CSS) et le code d'application (votre requête en PHP et autres).

Ainsi, plus besoin de se casser la tête à retrouver la requête dans la structure HTML du site ou encore à rechercher la variable dans une masse de texte.

Le rouage d'un moteur de templates est bien huilé. En effet, prenons une page d'index contenant une série de news. Le fichier PHP va se charger de rechercher les informations dans la base de données : titre, contenu, créateur…

Ce fichier ne contient aucune balise HTML ayant une influence sur l'affichage. Et d'un autre côté, on trouve un fichier de template (.html ou .html.twig) qui contient la structure HTML de la page web. Grâce au moteur de templates, les informations contenues dans le fichier PHP lui seront transmises et le moteur regroupera les deux fichiers en un seul par une compilation.

Voici comment fonctionne la compilation complète des deux fichiers :

1. Lecture du fichier PHP ;
2. Lecture du template ;
3. Compilation ;
4. Création du script PHP à partir des deux autres ;
5. Exécution du code généré.

Bien sûr, un tel système comprend son lot d'avantages, mais aussi d'inconvénients que nous allons voir tout de suite.

## 3. Avantages.

Comme je viens de le dire, l'utilisation d'un moteur de templates a de réels avantages dans la création d'un site web.

- La séparation des deux codes permet une meilleure visibilité dans le code.
- On peut alors toucher ou modifier un des deux fichiers sans que cela ait un impact sur l'autre.
- La mise en cache est proposée et permet ainsi d'économiser les ressources des serveurs avec la possibilité de configurer le tout.
- 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.

## 4. Inconvénients.

Cependant il comporte aussi son lot d'inconvénients.

- Son utilisation va retarder le chargement de votre page, mais il sera compensé en partie par Twig, un moteur de template assez rapide.
- Il faut étudier le langage du template.
- La lecture des erreurs est assez compliquée.

# 5. Documentation.

Le plus dur à retenir dans l'utilisation d'un moteur de templates ce sont principalement sa syntaxe, ses fonctions de base qu'il peut exécuter de lui-même sans avoir à les créer etc... C'est pour cela qu'une documentation en ligne est disponible. Je vous donne le lien, n'hésitez jamais pour aller la voir. Elle est très complète et explicite. (Seul bémol pour les anglophobes, celle-ci est entièrement en Anglais mais relativement facile à comprendre).

http://twig.sensiolabs.org/documentation

# II. Utilisation de Twig

## 1. Installation hors Symfony2.

Comme je le disais, il est tout à fait possible d'installer et d'utiliser Twig dans son projet sans pour autant que celui-ci soit développé avec Symfony2.

Pour cela, il faut télécharger l'archive sur le site de Twig, le décompresser dans le dossier de l'application web et dans chacune des pages du site ajouter une ligne de code.

L'archive est disponible à cette adresse : https://github.com/twigphp/Twig/tags

1. Choisissez la version que vous souhaitez de Twig.
2. Choisissez 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,
3. Décompressez l'archive,
4. Copiez les fichiers dans un dossier nommé « twig » dans le dossier de votre site.

Avant tout, il est plus pratique de créer un dossier nommé « templates » et de mettre tous les templates que vous allez créer dans ce dossier.

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).

```html+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
    ));
```

***Explication du code :***

Le code ici est assez simple. On inclut le fichier Autoloader.php. Ensuite, on indique le dossier 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.

Notez qu'ici n'est présent que l'option du cache mais il existe plusieurs options disponibles :

- cache : prend en argument le dossier où vous stockez les templates en cache ou bien false pour ne pas s'en servir. 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 il est à true.

Pour plus d'options je vous renvoie à cette section de la documentation :
http://twig.sensiolabs.org/doc/api.html#environment-options

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 :

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

## 2. Fonctionnalités (Affichage de variables, boucles etc…).

La particularité d'un moteur de templates c'est qu'il permet un affichage dynamique d'une page web par le biais de ses fonctionnalités.

Ainsi, il peut afficher des variables sur lesquelles il peut appliquer des filtres. Il peut aussi faire des boucles afin d'afficher un ensemble d'informations ou de variables. Tout cela sans PHP et uniquement en HTML, CSS et langage Twig.

Toujours dans le cas d'une série de news. Il pourra par exemple afficher le nom de la catégorie avec l'affichage d'une simple variable, lui appliquer un filtre pour que le nom de la catégorie soit en majuscule, puis, afficher toutes les news de cette catégorie avec le nom de l'auteur, la date de publication etc… grâce à une boucle.

# 3. Syntaxes.

Ce qu'il faut savoir avant de commencer l'apprentissage des syntaxes de Twig c'est que chacune des variables que l'on va afficher ou auxquelles on va appliquer des filtres ou sur lesquelles on va effectuer un traitement devra impérativement, au préalable, avoir été récupérée dans le contrôleur et envoyée à la vue sous forme d'un tableau.

On ne va pas lui envoyer les variables telles quelles mais des instances d'objet (par exemple une news ou une liste de news et non pas juste le nom de la news et son contenu).

Je mettrais à la fin du cours des tableaux des principales syntaxes.

Afin de faciliter la compréhension des syntaxes on va définir un petit contexte.

***Contexte :***

On souhaite afficher des news sur un site. Une news est une instance de l'objet News que l'on a auparavant initialisée. On aura récupéré les news et on les aura envoyés à la vue. En fonction de la syntaxe que je vais vous montrer on décidera soit d'afficher la liste des news soit le contenu d'une news.

## A. Affichage.

L'affichage d'une variable se fait relativement aisément. On écrit le nom de la variable que l'on souhaite afficher entre deux accolades.

Par exemple si on souhaite afficher uniquement le titre de la news on écrira :

```
1   {{ news.titre }}
```

***Remarque :***

Vous voyez que l'on a écrire « news.titre » et non pas juste « titre ». En effet, comme je l'ai dit précédemment on donne à la vue une instance de l'objet News. On est donc obliger d'écrire l'objet puis ce que l'on souhaite séparé d'un « . » comme un peut le faire en Programmation Orientée Objet.

### B. *Traitement.*

Un traitement comme une boucle par exemple s'effectue en écrivant la structure de contrôle entre deux % eux même entre deux accolades.

Par exemple si on souhaite afficher une liste (de titres) de news on écrira :

```
{% for news in listNews %}
    {{ news.titre }}
{% endfor %}
```

***Remarque :***

On a ici une boucle FOR. Dans Twig la boucle FOR sous cette forme est l'équivalent du FOREACH (ou POUR CHAQUE) que l'on peut utiliser en POO. On remarque aussi que le traitement doit absolument se terminer avec la ligne : {% end… %}. Pour l'affichage du titre on vient de le voir donc aucun problème de ce côté-là.

En revanche, reprenons la structure de contrôle : {% for news in listNews %}. Traduisons-la en pseudo-code : POUR CHAQUE news DANS listNews. Notre seul problème ici c'est listNews. Rappelez-vous ce que je vous ai dit tout à l'heure : « On aura récupéré les news et on les aura envoyés à la vue. » En effet, on a envoyé à la vue les news que l'on a récupéré seulement on ne va pas les envoyer une par une. On envoie donc une variable listNews qui contient un tableau avec toutes les instances de news que l'on a récupéré.

### C. *Tests.*

Les tests s'effectuent de la même manière que les deux points précédents. On va décider de les afficher ou alors de les utiliser dans un traitement.

Imaginons que les news aient une image mais que dans le cas où la news n'en a pas on souhaite afficher « Pas d'image pour cette news » on va donc faire le test pour savoir si la news contient une image et si elle n'en contient pas on n'affichera le message. Des tests sont déjà présents par défaut dans Twig pour le faire. Dans ce cas on utilisera le test : null

```
{% if news.image is null %}
    <p>Pas d'image pour cette news</p>
{% endif %}
```

***Remarque :***

On remarque tout de suite que le traitement est similaire au point précédent. Le code est suffisamment concis : SI la variable de l'image est null ALORS on affiche le texte. Remarquez aussi que l'on a du code HTML en effet les vues contiennent le code HTML et CSS du site.

### D. Opérateurs.

| Opérateur | Exemple d'utilisation |
|---|---|
| in | {% for news in listNews %} |
| is | {% if news.image is null %} |
| +, -, /, %, //, *, ** | Compteur = compteur +1 |
| ==, !=, <, >, >=, <=, ===, starts with, ends with, matches | {% if news.auteur == 'Dupont' %} |
| .., \|, ~, ., [], ?: | {{ news.titre\|title }} |

### E. Fonctions.

Les fonctions vont permettre de faire pleins de choses dans nos vues. On va par exemple pouvoir créer des blocks (nous verrons pourquoi par la suite), on va pouvoir inclure des fichiers, on va pouvoir utiliser le système d'héritage afin de modifier que certains éléments et garder la trame principale du site etc…

Pour utiliser une extension on utilise les deux syntaxes que l'on a vu précédemment. On va voir les deux cas possibles.

Imaginons que nous souhaitions inclure un template de commentaire dans une news :

```
1   {{ include('templateCommentaire.html.twig') }}
```

On voit alors que l'inclusion se fait avec la syntaxe {{ … }}. On utilise la fonction include (comme en php) et on lui passe en paramètre le nom du fichier à inclure.

Le design d'un site suit une trame principale puis en fonction de la page où on se trouve seul se petit morceaux du site change mais la trame reste la même. C'est dans ce cas que l'on utilise l'héritage (notion que l'on verra plus en détail dans la suite du cours). On a par conséquent un fichier de template principal qui va constituer la trame principale. Quand on arrive dans le template d'un bout de page il faut bien lui dire que l'on utilise une trame principale. Il faut donc lui dire quel template est son « parent » qu'on appelle généralement un layout :

```
1   {% extends "layout.html.twig" %}
```

La syntaxe est donc la suivante : {% extends 'nomdulayout' %}. Le mot clé extends permet l'héritage. En effet on n'inclut pas un template parent on le fait hériter de celui-ci.

### *F. Filtres.*

Les filtres vont permettre la modification rapide du contenu d'une variable. Par le titre d'une news par exemple « nouveautés technologiques de la semaine » si on lui appliquait le filtre title, lors de l'affichage le filtre se chargerait de mettre chaque première lettre du mot en majuscule ce qui donnerait : « Nouveautés Technologiques De La Semaine ».

```
1   {{ news.titre|capitalize }}
2   {{ news.auteur|title }}
```

La syntaxe est donc l'affichage d'une variable normalement séparée du nom du filtre par un pipe (tuyau : | (Alt Gr+6)).

Ici on aura le titre de la news avec la première lettre du premier mot en majuscule et le nom de l'auteur avec la première lettre de chacun des noms en majuscule (par exemple :
« Nouveautés technologiques de la semaine » « Jean Dupont »).

Si vous souhaitez mettre le même filtre sur plusieurs variables qui sont au même endroit, plutôt que de mettre un filtre sur chacune des variables, il est possible d'appliquer le filtre sur un bloc contenant les variables. On peut aussi mettre plusieurs filtres à la suite afin que chacun soit appliqué :

```
1   {% filter title|escape %}
2       {{ news.titre }}
3       {{ news.auteur }}
4   {% endfilter %}
```

Le titre de la news et le nom de l'auteur auront chacun la première lettre de chaque mots en majuscules et seront chacun échappés.

## *4. Héritage.*

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.

***Le principe :***

Le principe est simple : vous avez un template père qui contient le design de votre site ainsi que quelques trous (appelés « blocks » en anglais, que nous nommerons « blocs » en français) et des templates fils qui vont remplir ces blocs. Les fils vont donc venir hériter du père en remplaçant certains éléments par leur propre contenu.

L'avantage est que les templates fils peuvent modifier plusieurs blocs du template père. Avec la technique des include(), un template inclus ne pourra pas modifier le template père dans un autre endroit que là où il est inclus !

Les blocs classiques sont le centre de la page et le titre. Mais en fait, c'est à vous de les définir ; vous en ajouterez donc autant que vous voudrez.

Considérons que nous sommes dans un projet Symfony2 avec un bundle NewsBundle pour faire le travail dont on parle depuis tout à l'heure.

Prenons l'exemple tout simple d'un template père (communément appelé layout) que nous plaçons dans le dossier : src/MonSite/NewsBundle/Resources/views/layout.html.twig :

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>{% block title %}Mon site{% endblock %}</title>
  </head>
  <body>

    {% block body %}
    {% endblock %}

  </body>
</html>
```

***Explication :***

On a du code HTML standard avec les balises <html>, <head>, <meta />, <title> et <body>. Puis on découvre deux nouvelles parties : les parties {% block title %} et {% block body %}.

Ce sont les blocs dont je vous parlais juste au-dessus. Ce sont ces parties-là qui seront remplies par le template fils.

Et d'un template fils : src/MonSite/NewsBundle/Resources/views/News/news.html.twig :

```
{% extends "MonSiteNewsBundle::layout.html.twig" %}

{% block title %}{{ parent() }} - News{% endblock %}

{% block body %}

    {% for news in listNews %}
        {{ news.titre|capitalize }},
        <i>par {{ news.auteur|title }}, </i>
        le {{ news.date|date('d/m/Y') }}.
    {% endfor %}

{% endblock %}
```

***Qu'est-ce que l'on vient de faire ?***

Pour bien comprendre tous les concepts utilisés dans cet exemple très simple, détaillons un peu.

***Le nom du template père :***

On a placé ce template dans views/layout.html.twig et non dans views/qqch/layout.html.twig. C'est tout à fait possible ! En fait, il est inutile de mettre dans un sous-répertoire les templates qui ne concernent pas un contrôleur particulier et qui peuvent être réutilisés par plusieurs contrôleurs. Attention à la notation pour accéder à ce template : du coup, ce n'est plus MonSiteNewsBundle:MonController:layout.html.twig, mais MonSiteNewsBundle::layout.html.twig. C'est assez intuitif, en fait : on enlève juste la partie qui correspond au répertoire MonController. C'est ce que l'on a fait à la première ligne du template fils.

Pour le traitement, ici, on voit ce que l'on a déjà vu. On affiche la liste des news avec pour chacune le nom de l'auteur et la date.

***La balise {% block %} côté père :***

Pour définir un « trou » (dit bloc) dans le template père, nous avons utilisé la balise {% block %}. Un bloc doit avoir un nom afin que le template fils puisse modifier tel ou tel bloc de façon nominative. La base, c'est juste de faire {% block nom_du_block %}{% endblock %} et c'est ce que nous avons fait pour le body. Mais vous pouvez insérer un texte par défaut dans les blocs. C'est utile pour deux cas de figure :

- Lorsque le template fils ne redéfinit pas ce bloc. Plutôt que de n'avoir rien d'écrit, vous aurez cette valeur par défaut.
- Lorsque les templates fils veulent réutiliser une valeur commune. Par exemple, si vous souhaitez que le titre de toutes les pages de votre site commence par « Mon site », alors depuis les templates fils, vous pouvez utiliser {{ parent() }} qui permet d'utiliser le contenu par défaut du bloc côté père. Regardez, nous l'avons fait pour le titre dans le template fils.

***La balise {% block %} côté fils :***

Elle se définit exactement comme dans le template père, sauf que cette fois-ci on y met notre contenu. Mais étant donné que les blocs se définissent et se remplissent de la même façon, vous avez pu deviner qu'on peut hériter en cascade ! En effet, si l'on crée un troisième template petit-fils qui hérite de fils, on pourra faire beaucoup de choses.

***Le modèle « triple héritage » :***

Pour bien organiser ses templates, une bonne pratique est sortie du lot. Il s'agit de faire de l'héritage de templates sur trois niveaux, chacun des niveaux remplissant un rôle particulier. Les trois templates sont les suivants :

- Layout général : c'est le design de votre site, indépendamment de vos bundles. Il contient le header, le footer, etc. La structure de votre site donc (c'est notre template père).
- Layout du bundle : il hérite du layout général et contient les parties communes à toutes les pages d'un même bundle. Par exemple, pour notre plateforme d'annonce, on pourrait afficher un menu particulier, rajouter « Annonces » dans le titre, etc.
- Template de page : il hérite du layout du bundle et contient le contenu central de votre page.

**Puisque le layout général ne dépend pas d'un bundle en particulier, où le mettre ?**

Dans votre répertoire /app ! En effet, dans ce répertoire, vous pouvez toujours avoir des fichiers qui écrasent ceux des bundles ou bien des fichiers communs aux bundles. Le layout général de votre site fait partie de ces ressources communes. Son répertoire exact doit être app/Resources/views/layout.html.twig.

Et pour l'appeler depuis vos templates, la syntaxe est la suivante : « ::layout.html.twig ». Encore une fois, c'est très intuitif : après avoir enlevé le nom du contrôleur tout à l'heure, on enlève juste cette fois-ci le nom du bundle.

***L'inclusion de templates :***

***La théorie : Quand faire de l'inclusion ?***

Hériter, c'est bien, mais inclure, ce n'est pas mal non plus. Prenons un exemple pour bien faire la différence.

Le formulaire pour commenter une news est le même que celui pour… commenter une annonce par exemple. On ne va pas faire du copier-coller de code. C'est ici que l'inclusion de templates intervient. On a nos deux templates MonSiteNewsBundle:News:news.html.twig et MonSiteNewsBundle:Annonce:annonce.html.twig qui héritent chacun de MonSiteNewsBundle::layout.html.twig.

L'affichage exact de ces deux templates diffère un peu, mais chacun d'eux inclut MonSiteNewsBundle:News:form.html.twig à l'endroit exact pour afficher le formulaire.

On voit bien qu'on ne peut pas faire d'héritage sur le template form.html.twig, car il faudrait le faire hériter une fois de news.html.twig, une fois de annonce.html.twig, etc. Et si un jour, nous souhaitons ne le faire hériter de rien du tout pour afficher le formulaire tout seul dans une popup par exemple ? Bref, c'est bien une inclusion qu'il nous faut ici.

***La pratique : Comment le faire ?***

Cela se fait très facilement. Il faut utiliser la fonction {{ include() }}, comme ceci :

```
1  {% extends "MonSiteNewsBundle::layout.html.twig" %}
2
3  {% block title %}{{ parent() }} - News{% endblock %}
4
5  {% block body %}
6
7      {% for news in listNews %}
8          {{ news.titre|capitalize }},
9          <i>par {{ news.auteur|title }}, </i>
10         le {{ news.date|date('d/m/Y') }}.
11     {% endfor %}
12
13     <h2>Commentaires</h2>
14     {{ include("MonSiteNewsBundle::form.html.twig") }}
15
16 {% endblock %}
```

# 5. Tableau des principales syntaxes.

***Tableau des principales syntaxes d'affichage des variables :***

| Description | Exemple Twig | Équivalent PHP |
|---|---|---|
| Afficher une variable | Pseudo :<br>{{ pseudo }} | Pseudo : <?php echo $pseudo; ?> |
| Afficher l'index d'un tableau | Identifiant :<br>{{ user['id'] }} | Identifiant : <?php echo $user['id']; ?> |
| Afficher l'attribut d'un objet, dont le getter respecte la convention $objet->getAttribut() | Identifiant :<br>{{ user.id }} | Identifiant : <?php echo $user->getId(); ?> |
| Concaténer | Identité :<br>{{ nom ~ " " ~ prenom }} | Identité : <?php echo $nom.' '.$prenom; ?> |

***Tableau des principaux traitements utiles :***

| Traitement | Description | Syntaxe |
|---|---|---|
| Extends | Fait hériter le template d'un template père. | {% extends "MonSiteNewsBundle::layout.html.twig" %} |
| For (Pour) | Permet de faire une boucle : POUR, POUR CHAQUE | {% for i in 0..10 %} * {{ i }} {% endfor %};<br>{% for … in … %} … {% endfor %};<br>{% for … in … if … %} … {% endfor %} ;<br>{% for … in … %} … {% else %} … {% endfor %} |
| If (Si) | Permet de faire une structure de contrôle. | {% if … %} … {% endif %};<br>{% if … %} … {% elseif … %} … {% else %} …<br>{% endif %} |

***Tableau des principales fonctions utiles :***

| Fonctions | Description | Syntaxe |
|---|---|---|
| Block | Définit un bloc pour l'héritage. | {% block body %} ... {% endblock %} |
| Include | Inclut un fichier. | {{ include 'nomDuFichier' }} |
| Parent | Affiche le contenu du template père. | {{ parent() }} |
| Max/Min | Affiche la valeur max / min d'une liste. | {{ max(1, 3, 2) }} / {{ min(1, 3, 2) }} |
| Path | Permet d'accéder à l'URL de la route. | {{ path('nomDeLaRoute') }} |

***Tableau des principaux filtres utiles :***

| Filtre | Description | Exemple Twig |
|---|---|---|
| Upper | Met toutes les lettres en majuscules. | {{ var\|upper }} |
| Striptags | Supprime toutes les balises XML. | {{ var\|striptags }} |
| Date | Formate la date selon le format donné en argument. La variable en entrée doit être une instance de Datetime. | {{ date\|date('d/m/Y') }}<br>Date d'aujourd'hui :<br>{{ "now"\|date('d/m/Y') }} |
| Format | Insère des variables dans un texte, équivalent à printf. | {{ "Il y a %s pommes et %s poires"\|format(153, nb_poires) }} |
| Length | Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne. | Longueur de la variable :<br>{{ texte\|length }}<br>Nombre d'éléments du tableau :<br>{{ tableau\|length }} |
| Lower | Met toutes les lettres en minuscule. | {{ var\|lower }} |
| Title | Met la première lettre de chaque mot en majuscule. | {{ var\|title }} |
| Capitalize | Met la première lettre du premier mot en majuscule. | {{ var\|capitalize }} |

***Tableau des principaux tests utiles :***

| Test | Description | Syntaxe |
|---|---|---|
| Defined | Vérifie si une variable est définie dans le contexte actuel. | {% if … is defined %} … {% endif %};<br>{% if foo.bar is defined %}<br>… {% endif %} ;<br>{% if …['…'] is defined %}<br>… {% endif %} |
| Divisbleby | Vérifie si une variable est divisible par un nombre. | {% if … is divisible by(3) %}<br>… {% endif %} |
| Empty | Vérifie si une variable est vide. | {% if … is empty %} … {% endif %} |
| Null | Retourne true si la variable est null. | {% var is null %} … {% endif %} |
| Sameas | Vérifie si une variable est la même que celle d'une autre variable. C'est l'équivalent de === en PHP. | {% if … is same as(false) %}<br>… {% endif %} |