

# Le moteur de template Twig

# Table des matières

<b>I. Comprendre Twig et son installation</b>	<b>3</b>
A. Symfony, Composer et environnement de travail .....	3
B. Les modalités de Twig.....	3
<b>II. Exercice : Quiz</b>	<b>5</b>
<b>III. Utilisation et syntaxe de Twig</b>	<b>6</b>
A. Les Variables .....	6
B. Les filtres.....	8
C. Les fonctions .....	9
D. Les conditions et les tests .....	10
E. Boucle for.....	11
F. L'héritage.....	12
<b>IV. Exercice : Quiz</b>	<b>13</b>
<b>V. Essentiel</b>	<b>14</b>
<b>VI. Auto-évaluation</b>	<b>14</b>
A. Exercice .....	14
B. Test.....	14
<b>Solutions des exercices</b>	<b>15</b>

# I. Comprendre Twig et son installation

**Durée :** 1 h

**Pré requis :** notion de PHP et notion de POO

## Contexte

Vous commencez maintenant à comprendre les avantages d'un framework HTTP comme Symfony ainsi que l'architecture MVC (Modèle-Vue-Contrôleur) qui est aujourd'hui omniprésente dans la POO (Programmation Orientée Objet) pour les applications web modernes. Si vous suivez cette formation, vous avez vu le rôle essentiel du contrôleur et son utilité qui consiste à renvoyer le plus souvent une vue pour une application web.

Mais qu'est-ce qu'une vue ? La vue dans une application web c'est tout ce qui est visible à l'écran. Cela peut être l'affichage d'une page avec du texte, des images, des vidéos, mais aussi un moyen de communication avec l'utilisateur, avec des zones de saisie, des formulaires, des boutons et tout ce qui propose une interaction. Cette partie visuelle offre un vrai design et donc une interface.

Dans le monde du développement, on divise en deux parties une application web entre le front end et le back end. Le front étant la partie visible et le back la partie cachée. Symfony offre une architecture structurale qui permet à chacune de ces deux parties de pouvoir travailler séparément (même si dans la réalité c'est plus complexe et que souvent il est nécessaire de travailler en full stack). C'est là qu'intervient Twig, le moteur de template généralement utilisé avec Symfony. Celui-ci va permettre de développer du front sans pour autant avoir besoin de connaître le PHP et avoir à toucher à l'ensemble du code de l'application.

Mais qu'est-ce qu'est exactement un moteur de template ? Quel est son rôle ? Comment fonctionne-t-il ? Voici quelques-unes des questions auxquelles nous allons répondre à travers ce cours.

## A. Symfony, Composer et environnement de travail

Avant de pouvoir s'atteler à comprendre ce qu'est un moteur de template et comment l'installer, il est important de comprendre qu'avec Symfony 5.4, la version de Symfony préconisée dans ce cours, il faut installer Composer. Composer est un outil de gestion de dépendances qui permet aux développeurs de gérer les librairies et les dépendances de leur projet de manière simple et efficace. Il a été créé pour résoudre les problèmes courants liés à la gestion des dépendances dans les projets PHP, et est devenu un outil incontournable pour les développeurs PHP. Symfony a contribué à l'amélioration de Composer en ajoutant des recettes, qui sont des scripts PHP utilisés pour simplifier l'installation et l'intégration de certaines dépendances, comme Twig.

Si vous n'avez pas installé Composer, veuillez vous référer au site officiel de Composer<sup>1</sup> afin de suivre la procédure d'installation car il vous sera nécessaire.

Vous pouvez choisir bien sûr l'IDE (environnement de développement intégré) que vous souhaitez. Par exemple, Visual Studio Basic vous permettra d'appeler facilement le terminal et de pointer immédiatement sur le bon dossier.

## B. Les modalités de Twig

### Qu'est-ce qu'un moteur de templates ?

Le rôle d'un moteur de templates est principalement de vous aider dans la lisibilité et la logique d'une application. Ce que fait précisément un moteur de templates, c'est de rassembler le code pour l'affichage (HTML, CSS et Javascript voir Sass) avec le code PHP de l'application.

Imaginez que vous avez un modèle de lettre standard que vous utilisez pour écrire des lettres à différentes personnes. Ce modèle contient des variables telles que le nom de la personne à qui la lettre est destinée, l'adresse de cette personne et le contenu de la lettre. Chaque fois que vous voulez écrire une lettre, vous remplissez ces variables avec les valeurs appropriées et le modèle génère une lettre personnalisée. C'est un peu comme si vous

---

<sup>1</sup> <https://getcomposer.org/>

aviez un moteur de templates pour générer des lettres. En informatique, un moteur de templates fonctionne de manière similaire en remplaçant les variables prédéfinies dans un modèle de texte par leurs valeurs pour générer du contenu.

Ce que l'on souhaite, c'est de pouvoir à la fois séparer le HTML du code PHP, et à la fois pouvoir générer du code dynamique et insérer quelques conditions et variables issues de la base de données.

C'est tout l'intérêt d'un moteur de templates comme Twig qui va permettre de regrouper dans une page le HTML afin de ne pas avoir des centaines de balises HTML qui gêneraient la lisibilité du code PHP ainsi que l'architecture de la POO qui consiste à tout transformer en objet. Mais aussi de pouvoir intégrer grâce à une syntaxe simple, qui ne demande pas de connaître le PHP, quelques conditions et données.

## Méthode Installation de Twig

Si vous avez utilisé le mode d'installation d'application web de Symfony, il est fort probable que Twig soit déjà installé par défaut. Mais cela n'est pas une obligation car Twig est un moteur de templates utilisé par Symfony mais ne fait pas partie de Symfony. Cela signifie que vous pouvez installer Twig dans un projet PHP sans Symfony mais aussi dans des projets Open Source comme Drupal8, eZPublish, phpBB, Matamo, OroCRM ainsi que sur de nombreux frameworks tels que Lavarel, même si, par défaut, ils utilisent d'autres moteurs de templates.

Voici comment l'installer avec Composer, il suffit de taper cette ligne de commande dans le répertoire de votre projet :

```
1 $ composer require "twig/twig:^3.0"
```

Cela va télécharger et installer Twig et toutes les dépendances nécessaires dans votre projet. Vous devrez peut-être utiliser sudo si vous rencontrez des erreurs de permission. Par défaut un projet Symfony inclut l'autoloader de Composer, il ne sera pas nécessaire de require\_once dans vos pages Twig.

Vous pouvez ensuite utiliser Twig dans votre code en suivant la documentation de Twig : [Symfony Hub](https://twig.symfony.com/doc/2.x/)<sup>1</sup>

Veuillez noter que la version 3.x de Twig demande une version de PHP 7.2.5 au minimum pour fonctionner.

## Avantages, inconvénient de Twig

Les avantages :

- **Clarté** : grâce à la syntaxe facile de Twig, il n'est pas nécessaire d'écrire du PHP dans du HTML car il est possible d'appeler des variables, de disposer de structures de contrôle ou encore de fonctions PHP. La séparation des deux codes permet une meilleure visibilité et compréhension.
- **Documentation** : la documentation officielle de sensiolabs est claire, fournie et bien expliquée ainsi que régulièrement mise à jour : [Symfony Hub](https://twig.symfony.com/doc/)<sup>2</sup>.
- **Rapide** : Twig compile les templates et utilise une optimisation du code PHP, ainsi, les performances sont optimisées par rapport au code PHP ordinaire. La mise en cache qui est paramétrable permet d'économiser les ressources des serveurs.
- **Sécurité** : Twig dispose d'une sandbox pour évaluer le code non fiable. Ainsi, les variables sont sécurisées automatiquement par un filtre, il n'est donc pas nécessaire de les protéger en amont.
- **Flexibilité** : Twig est alimenté par un lexer (c'est un analyseur syntaxique ou programme qui se charge d'analyser et d'extraire ou isolé des éléments d'une grammaire) et un parser. Cela permet de définir et personnaliser ses propres balises et filtres et de créer son propre DSL (langage dédié).

<sup>1</sup> <https://twig.symfony.com/doc/2.x/>

<sup>2</sup> <https://twig.symfony.com/doc/>

Les inconvénients :

- Dépendance : Twig nécessite l'installation de Composer et de certaines dépendances, ce qui peut être un problème pour certains projets.
- Twig est un moteur de templates performants et peut faire gagner en performances de par son architecture mais de façon générale un moteur de templates fait retarder le chargement des pages.
- Twig possède une syntaxe simple mais demande malgré tout un petit apprentissage et donc du temps.
- Twig, de par sa simplicité, n'a pas autant de flexibilité (DOM virtuel, asynchrone, fragmentation/états des éléments) qu'un langage front à part entière type React, Vue, etc.

### Intégration IDE

De nombreux IDE prennent en charge la complétion automatique du Twig ainsi que la mise en surbrillance de la syntaxe afin d'optimiser l'environnement de travail. Citons les IDE ainsi que l'extension nécessaire pour cette prise en charge :

- Atom avec « *PHP-twig for atom* »
- Coda 2 avec « *Twig syntax mode* »
- Coda et SubEthaEdit avec « *Twig language definition* »
- Eclipse avec « *Twig plugin* »
- Emacs avec « *web-mode.el* »
- GtkSourceView avec « *Twig language definition* »
- Komodo et Komodo Edit avec « *Twig highlight/syntax check mode* »
- Netbeans avec « *Twig syntax plugin* »
- Notepad++ avec « *Notepad++ Twig highlighter* »
- PhpStorm natif depuis la version 2.1
- Sublime Text avec « *Twig bundle* »
- Textmate avec « *Twig bundle* »
- Vim avec « *Jinja syntax plugin* »
- Visual Studio Code avec « *twig pack* »

Vous pouvez aussi effectuer des tests en ligne grâce au Playground twigfiddle sur TWIGFiddle<sup>1</sup>.

### Exercice : Quiz

[solution n°1 p.17]

#### Question 1

Il n'y a pas besoin d'installer Twig, il est présent par défaut dans un projet Symfony.

- ☐ Vrai
- ☐ Faux

#### Question 2

---

<sup>1</sup> <https://twigfiddle.com/>

Les pages Twig sont généralement traitées par les développeurs :

- ☐ Front end
- ☐ Back end
- ☐ Full stack

Question 3

Qu'est-ce qu'un IDE en français ?

- ☐ Un intégrateur de dépôt environnemental
- ☐ Un environnement de développement intégré
- ☐ Un éditeur de code source

Question 4

Il est possible de personnaliser ses balises dans Twig.

- ☐ Vrai
- ☐ Faux

Question 5

Il est nécessaire d'être sous Symfony pour utiliser Twig.

- ☐ Vrai
- ☐ Faux

### III. Utilisation et syntaxe de Twig

Twig peut générer du texte de n'importe quel format textuel (HTML, XML, CSV, Latex, etc.). Dans ce cours, nous partirons sur des exemples basés sur le HTML, qui représentent la plus grande utilisation de Twig.

Comme nous l'avons vu précédemment, la force d'un moteur de templates comme Twig se situe dans le fait de pouvoir insérer des variables, des expressions et des conditions PHP dans du HTML. Notez la syntaxe particulière qui est nativement supportée par Twig afin de nous apporter cette puissance :

- « `{{ ... }}` » : permet l'affichage d'une expression
- « `{% ... %}` » : exécute une action
- « `{# ... #}` » : permet d'ajouter des commentaires

#### A. Les Variables

La puissance de Twig c'est de pouvoir afficher des éléments issus de PHP. Par exemple un objet, un tableau ou, bien souvent, des éléments issus de la base de données. C'est le contrôleur qui a pour mission d'associer ces éléments à la vue. Pour Twig ces éléments sont des variables (attention à ne pas confondre avec une variable en PHP).

Ces variables peuvent être un objet et ce même objet peut avoir des attributs. On peut par ailleurs y accéder en utilisant la syntaxe :

```
{{ objet.attribut }}
```

Dans l'exemple ci-dessous « *image.name* » fait référence au nom de l'objet image. Twig affichera donc son nom. Grâce à cette syntaxe, il est possible d'accéder à des méthodes, des propriétés d'un objet, ou même un élément de tableau.

**Exemple**

Imaginons que notre image se nomme « *Belle Image* » et que l'on retrouve ceci dans le HTML :

```
1 <h1> Le nom de l'image est : {{ image.name }} </h1>
```

Dans cet exemple, Twig affichera toute la balise H1 dont ce qui se trouve entre les accolades, c'est-à-dire la variable image qui a pour attribut name. Voici le rendu :

« **Le nom de l'image est : Belle Image** ».

**La balise set**

Vous pouvez aussi à tout moment créer une variable grâce à la balise « **set** » :

**Exemple**

```
{% set exemple = 'coucou' %}
```

Cela définira la variable exemple avec la chaîne de caractères « *coucou* ». Vous pouvez ensuite utiliser cette variable ailleurs dans votre modèle en la référençant avec {{ exemple }}.

```
{% set exempleTableauIteratif = [1,2] %}
```

Ce code définira la variable exempleTableauIteratif comme étant un tableau contenant les éléments 1 et 2. Il ne produira aucune sortie dans le modèle.

```
{% set exempleTableauAssociatif = {'lundi' : 'travail'} %}
```

Ce code définira la variable exempleTableauAssociatif comme étant un tableau associatif (également appelé dictionnaire ou map) avec un seul élément. L'élément a une clé « *lundi* » et une valeur « *travail* ».

**Les variables globales**

Une variable globale est une variable déclarée à l'extérieur des classes mais pouvant être utilisée n'importe où dans le projet. Elle peut également prendre le nom de variable de portée globale.

Dans Symfony, pour créer des variables globales, c'est-à-dire des variables qui seront utilisables dans toute l'application sans avoir besoin de l'injecter à chaque fois dans le contrôleur, il faut commencer par se rendre dans le fichier twig.yaml qui se trouve dans le dossier config et créer sa variable globale.

Le chemin exact est : config/packages/twig.yaml.

**Exemple**

```
1 Twig:
2 default_path: '%kernel.project_dir%/templates'
3 globals:
4 titre: 'le titre que je veux réutiliser'
```

Ici, le chemin par défaut pour les modèles Twig est spécifié en utilisant la clé « *default path* » sous la section Twig du fichier de configuration services.yaml. La valeur de « *default path* » est une chaîne de caractères qui contient le marqueur de nom de répertoire de projet suivi du sous-répertoire /templates. Le marqueur de nom %kernel.project\_dir% est une référence au répertoire de projet dans l'environnement de programmation ou l'application.

En déclarant dans le fichier twig.yaml comme dans l'exemple ci-dessus la variable « *titre* » dans globals, il est possible d'appeler celle-ci n'importe où dans un fichier Twig avec la déclaration suivante :

```
<h1> {{ titre }} </h1>
```

Twig fournit aussi certaines variables globales, en voici la liste avec une succincte description :

- debug : booléen indiquant si le débogage est activé ou non
- environment : indique l'environnement en cours, comme dev ou prod
- request : instance de `Symfony\Component\HttpFoundation\Request`
- session : instance de `Symfony\Component\HttpFoundation\Session`
- flashes : tableau de Session Flash message
- user : instance de `App\Entity\User`
- tokenStorage : instance de `Symfony UsageTrackingTokenStorage`

On appelle ces variables avec le préfixe « *app.* ».

#### Exemple

```
{% if not app.user %}
```

La boucle if avec not app.user vérifie si l'objet app.user est « *faux* », c'est-à-dire s'il vaut null, false, ou une chaîne vide. Si c'est le cas, le contenu de la boucle if sera exécuté.

## B. Les filtres

Les filtres permettent de modifier des variables. On sépare la variable du filtre avec le symbole pipe « | ».

Il est possible d'enchaîner plusieurs filtres, il faudra à chaque fois rajouter un pipe pour chaque filtre.

Sachez que vous pouvez créer vous-mêmes vos propres filtres selon vos besoins. Un filtre = une fonction d'une classe PHP appelée qui recevra vos différents paramètres.

#### Exemple Raw et length

Cette expression Twig affiche la longueur de la chaîne de caractères stockée dans la variable titre :

```
{{ titre | raw | length > 50 }}
```

Elle utilise également deux filtres : raw et length.

Le filtre raw permet de désactiver l'échappement des caractères HTML dans la chaîne de caractères. Cela signifie que les balises HTML dans la chaîne seront interprétées et affichées comme du code HTML au lieu d'être affichées comme du texte brut.

Le filtre length renvoie la longueur de la chaîne de caractères en nombre de caractères. L'opérateur de comparaison > vérifie si la longueur de la chaîne de caractères est supérieure à 50. Si c'est le cas, l'expression renvoie true, sinon elle renvoie false.

#### Exemple Parenthèse autour des arguments

Les filtres qui acceptent les arguments ont quant à eux des parenthèses autour des arguments comme ceci :

```
{{ texte | join(', ') }}
```

Cette expression Twig utilise le filtre join pour concaténer tous les éléments d'une liste en une seule chaîne de caractères.

Le filtre join prend en argument une chaîne de caractères qui sera utilisée comme séparateur entre chaque élément de la liste. Dans l'expression {{ texte | join(', ') }, la chaîne « , » (une virgule et un espace) est utilisée comme séparateur.



**Exemple** Apply et endapply

On peut notamment appliquer un filtre sur une section de code, avec « *apply* » et « *endapply* » comme suit :

```
{% apply upper%
```

Ce texte sera en majuscule

```
{% endapply %}
```

La balise `apply` avec le filtre `upper` permet de transformer tous les caractères d'une chaîne de caractères en majuscules. Dans cet exemple, la chaîne de caractères « *Ce texte sera en majuscule* » sera transformée en « *CE TEXTE SERA EN MAJUSCULE* » grâce au filtre `upper`.

La balise `apply` peut être utilisée avec n'importe quel filtre Twig. Elle permet d'appliquer un filtre à tout le contenu compris entre les balises `apply` et `endapply`.

**Remarque**

Il existe à ce jour plus d'une cinquantaine de filtres prédéfinis par Twig. Vous pouvez les retrouver dans la documentation officielle de Twig, mais vous avez aussi la possibilité de créer un filtre personnalisé. Pour cela, vous devrez créer une classe extension afin de pouvoir l'utiliser.

**C. Les fonctions**

Des fonctions peuvent être appelées pour générer du contenu. Les fonctions sont appelées par leur nom suivi de parenthèses « `()` » et peuvent avoir des arguments.

Twig fournit de nombreuses fonctions (leur nombre est fluctuant suivant la version de Twig que vous avez installée). En voici quelques-unes fournies avec Twig 3 :

- `attribute()` permet d'accéder aux attributs dynamiques d'une variable,
- `block()` permet de définir un bloc comme gabarit en utilisant l'héritage et pouvant être appelé plusieurs fois,
- `constant()` permet de retourner la valeur de la constante par la chaîne de caractères spécifiés,
- `cycle()` permet de retourner la valeur à la position spécifiée de cellule d'un tableau,
- `date()` permet de retourner la date avec le format spécifié,
- `dump()` permet de sortir le contenu d'une variable spécifiée,
- `include()` permet de retourner le rendu du contenu d'un gabarit,
- `max()` permet de retourner la plus grande valeur d'une séquence ou d'un ensemble de variables,
- `min()` permet de retourner la plus petite valeur d'une séquence ou d'un ensemble de variables,
- `parent()` permet de retourner le contenu du bloc parent quand l'héritage de gabarit est utilisé,
- `random()` permet de retourner une valeur aléatoire en fonction du contexte spécifié,
- `range()` permet de retourner la liste de nombres compris dans l'intervalle spécifié,
- `source()` permet de retourner le contenu d'un gabarit sans son rendu,
- `template_from_string()` permet de charger un gabarit à partir d'une chaîne de caractères spécifiée.

**Exemple** Utilisation d'une de ces fonctions

```
{{ max(1, 2, 3) }} ou {{ max([1, 2, 3]) }}
```

Cette expression Twig utilise la fonction `max` pour renvoyer le plus grand nombre dans une liste de nombres.

La fonction `max` prend en argument une liste de nombres et renvoie le nombre le plus élevé. Dans l'expression `{{ max(1, 2, 3) }}`, la liste de nombres est 1, 2, 3. La fonction `max` renverra donc 3.

## D. Les conditions et les tests

L'utilisation des conditions s'avère être particulièrement facile à manipuler avec Twig. Cela peut se faire avec **if**, **elseif**, **else**, **endif**. Ainsi, chaque structure de contrôle commence par une syntaxe ouvrante `{% if %}` puis fermante obligatoire `{% endif %}` comme le montre l'exemple ci-dessous.

### Exemple

```
{% if mvariable <= 10 %}
```

```
Je suis au niveau 1
```

```
{% elseif mvariable >10 and mvariable <20 %}
```

```
Je suis au niveau 2
```

```
{% else %}
```

```
Je suis au niveau 3
```

```
{% endif %}
```

Ce code utilise une boucle if avec plusieurs conditions pour afficher un message différent en fonction de la valeur de la variable mvariable.

La boucle if vérifie d'abord si la valeur de mvariable est inférieure ou égale à 10. Si c'est le cas, le texte « *Je suis au niveau 1* » sera affiché.

Si la valeur de mvariable n'est pas inférieure ou égale à 10, la boucle elseif vérifie si la valeur de mvariable est supérieure à 10 et inférieure à 20. Si c'est le cas, le texte « *Je suis au niveau 2* » sera affiché.

Si aucune des conditions précédentes n'est remplie, la boucle else s'exécutera et affichera le texte « *Je suis au niveau 3* ».

Il est possible de mettre dans les expressions des conditions, des opérateurs, des tests définis et même parfois de les combiner.

Voici la liste des types d'opérateurs utilisables dans les conditions ainsi que les tests proposés par le moteur de template Twig :

- Opérateurs
  - « `+` `-` `*` `/` `%` », mathématiques
  - « `and` `or` `not` », logiques
  - « `==` `!=` `<=` `>=` `<` `>` », comparaison
  - « `in` », booléen (ex : a in b est vrai si a est dans b)
  - « `is not` », test
  - « `..` », création de séquence
  - « `|` », filtre
  - « `~` », concaténation
  - « `?:` », ternaire
- Tests
  - « `constant` », vérifie qu'une variable a la même valeur qu'une constante globale ou qu'une classe,
  - « `defined` », vrai si la variable est définie dans le contexte courant,
  - « `divisible by` », vrai si la variable est divisible par un nombre donné en paramètre,
  - « `empty` », vrai si la variable est vide,

- « *null* », vrai si la variable est à null,
- « *even / odd* », vrai si la variable est paire/impair,
- « *Iterable* », vrai s'il est possible d'itérer sur la variable,
- « *Same as* », vrai si une variable est à l'identique.

**Exemple**

```
{% if mavariable is not empty %}
```

Dans cet exemple, il y a une combinaison entre « **is not** » et « **empty** ». La boucle if avec mavariable is not empty vérifie si la variable mavariable n'est pas vide. Si c'est le cas, le contenu de la boucle if sera exécuté. Une variable est considérée comme vide si elle vaut null, false, ou une chaîne de caractères vide.

**E. Boucle for**

Il est très facile de boucler chaque élément dans une séquence. Il suffit de déclarer celle-ci avec l'expression « **for** » et « **in** » et de terminer la boucle avec l'expression « **endfor** ».

**Exemple**

Voici quelques exemples :

```
{% for image in images %}
```

```
<li>{{ image.name }}</li>
```

```
{% endfor %}
```

Ce code utilise une boucle for pour afficher une liste de noms d'images.

La boucle for parcourt chaque élément de la liste images et exécute le contenu de la boucle pour chaque élément. Dans chaque itération de la boucle, l'élément en cours est disponible via la variable image.

Dans cet exemple, la boucle for affiche chaque nom d'image dans une balise « *li* ». La propriété name de l'objet image est utilisée pour récupérer le nom de l'image.

```
{% for i in 0..10 %}
```

```
{{ i }}
```

```
{% endfor %}
```

Ce code utilise une boucle for pour afficher tous les nombres de 0 à 10.

La boucle for parcourt chaque nombre de la plage de nombres spécifiée (0 à 10) et exécute le contenu de la boucle pour chaque nombre. Dans chaque itération de la boucle, le nombre en cours est disponible via la variable i.

Dans cet exemple, la boucle for affiche chaque nombre de la plage de 0 à 10.

Ou encore, pour la gestion des index de valeurs qui arrivent souvent :

```
{% for i in 0..10 %}
```

```
{{ i }}
```

```
{% endfor %}
```

## Les variables spéciales

À l'intérieur d'un bloc de boucle « *for* », il est possible d'accéder à certaines variables spéciales :

- `loop.index` : itération actuelle de la boucle (1 indexé)
- `loop.index0` : itération actuelle de la boucle (0 indexé)
- `loop.revindex` : nombre d'itérations à partir de la fin de la boucle (1 indexé)
- `loop.revindex0` : nombre d'itérations à partir de la fin de la boucle (0 indexé)
- `loop.first` : vrai si première itération
- `loop.last` : vrai si dernière itération
- `loop.length` : nombre d'éléments de la séquence
- `loop.parent` : contexte du parent

### Exemple

```
{% if loop.index == 1 %}
```

La boucle `if` avec `loop.index == 1` vérifie si l'index de la boucle en cours est égal à 1. Si c'est le cas, le contenu de la boucle `if` sera exécuté.

## F. L'héritage

L'héritage est une partie particulièrement intéressante de Twig. Il est possible de créer un modèle ou un squelette qui contient tous les éléments communs aux différentes pages de l'application. Pour cela, il faut créer des blocs et leur donner un nom.

### Exemple

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <link rel="stylesheet" href="style.css"/>
5     <title>{% block title %}{% endblock %}</title>
6   </head>
7   <body>
8     <div>{% block content %}{% endblock %}</div>
9     <footer>Bienvenue sur mon site internet</footer>
10  </body>
11 </html>
```

Ce code définit un modèle de page HTML qui utilise des blocs Twig pour personnaliser le titre et le contenu de la page.

Les balises `{% block %}` et `{% endblock %}` délimitent un bloc qui peut être remplacé par des blocs similaires des templates enfants qui étendent ce même modèle.

Dans cet exemple, le modèle a deux blocs : `title` et `content`. Le titre de la page sera remplacé par le contenu du bloc `title` dans les templates enfants, et le contenu principal de la page sera remplacé par le contenu du bloc `content` dans les templates enfants.

Grâce à l'héritage, une page enfant pourra être constituée ainsi :

```
1 {% extends "base.html.twig"%}  
2 {% block title %} Page enfant {% endblock%}  
3 {% block content %}  
4     <h1>Exemple</h1>  
5     {{ form(form) }}  
6 {% endblock%}
```

Ce code définit un template enfant qui étend le modèle base.html.twig et remplace les blocs title et content du modèle parent.

Le template enfant utilise la fonction form de Twig pour afficher un formulaire HTML généré à partir de l'objet form bien sûr si le composant de Form de Symfony est également installé.

```
1 $ composer require symfony/form
```

## Exercice : Quiz

[solution n°2 p.18]

### Question 1

Quel symbole opérateur n'est pas un opérateur mathématique ?

- ☐ \*
- ☐ /
- ☐ ==
- ☐ %

### Question 2

Quel pourrait être le résultat de cette fonction Twig : `{{ random(17) }}` ?

- ☐ 20
- ☐ 1
- ☐ 16,33

### Question 3

Une variable Twig est différente d'une variable PHP.

- ☐ Vrai
- ☐ Faux

### Question 4

Si `foo = 'hello'` et `bar = 'world'`, quelle est la bonne concaténation pour obtenir « *hello world* » ?

- ☐ `{{ foo|upper ~ ' ' ~ bar ~ '!' }}`
- ☐ `{{ foo ~ bar|upper ~ ! }}`
- ☐ `{{ 'foo ~ bar ~ !' }}`

### Question 5

{% extends "base.html.twig%" est la bonne façon pour utiliser l'héritage dans une page html enfant.

- ☐ Vrai
- ☐ Faux

## V. Essentiel

Un moteur de templates comme Twig est un vrai plus, car il vous permet d'afficher la vue de votre HTML tout en étant capable d'insérer des éléments issus de PHP que l'on appelle variable. Une variable pour Twig peut être un objet, un tableau, une variable PHP voire bien d'autres choses encore.

Twig offre une syntaxe particulièrement facile à apprendre et propose des fonctionnalités comme la boucle for, les conditions, etc., qui permettent une utilisation dynamique du code.

Dans ce cours nous avons pu découvrir les avantages d'utiliser un moteur de templates et comprendre pourquoi Symfony propose Twig par défaut. Nous avons découvert les variables, qu'elles soient simples, globales ou spéciales. Mais aussi l'utilisation de filtres, de quelques fonctions intégrées à ce moteur de templates, des boucles, des conditions et enfin de l'héritage.

Grâce à un moteur de templates, il est plus facile de séparer les métiers, ainsi, un développeur front end qui ne maîtrise pas le PHP pourra travailler de son côté à construire une interface utilisateur ergonomique et agréable.

Il est vrai qu'un moteur de templates comporte quelques inconvénients et nous l'avons évoqué, mais Twig apporte un vrai plus pour une application web complexe.

## VI. Auto-évaluation

### A. Exercice

Vous êtes un développeur front end expérimenté dans une agence web où vous gérez de nombreuses applications web. On vous demande, pour les applications web créées avec Symfony de gérer tous les dossiers templates qui contiennent toutes les pages html.twig.

#### Question 1

[solution n°3 p.19]

Votre projet 1 contient de nombreuses pages et les récurrences sont très nombreuses mais contiennent parfois des nuances. En quoi une page modèle vous sera très utile.

#### Question 2

[solution n°4 p.19]

Dans votre projet 2, vous serez amené à utiliser fréquemment certains éléments de bootstrap comme les badges et adapter leur couleur. Que pourriez-vous créer qui vous fera gagner beaucoup de temps ? Expliquez pourquoi.

### B. Test

#### Exercice 1 : Quiz

[solution n°5 p.19]

Question 1

Il est possible de faire un dump avec Twig.

- ☐ Vrai
- ☐ Faux

Question 2

Quelle variable n'est pas une variable globale native en Twig ?

- ☐ test
- ☐ user
- ☐ request

#### Question 3

Plus tôt, une variable « *montant* » a été déclarée. Quelle condition va permettre d'obtenir un résultat si la valeur de « *montant* » est supérieure à 20 ?

- ☐ {% if montant > 20 %}
- ☐ {{% if montant > 20 %}}
- ☐ {% if montant < 20 %}

#### Question 4

Si cette ligne de code est dans la page parent : `{% block header_img%} {% endblock %}`, il me suffira de l'appeler et d'ajouter mon image dans le bloc pour modifier l'image.

- ☐ Vrai
- ☐ Faux

#### Question 5

Pour récupérer tous les éléments d'un tableau, je peux faire une boucle foreach en Twig.

- ☐ Vrai
- ☐ Faux

## Solutions des exercices






**Exercice p. 5 Solution n°1****Question 1**

Il n'y a pas besoin d'installer Twig, il est présent par défaut dans un projet Symfony.

☐ Vrai

☒ Faux

 Tout dépend du mode d'installation du projet Symfony que vous avez choisi, il peut être nécessaire de l'ajouter.


**Question 2**

Les pages Twig sont généralement traitées par les développeurs :

☒ Front end

☐ Back end

☐ Full stack

 En théorie, ce sont les développeurs front end qui doivent s'occuper de développer ces pages, mais dans la pratique, et surtout dans les petites structures, un développeur back end aura tendance à s'occuper également de cette partie.


**Question 3**

Qu'est-ce qu'un IDE en français ?

☐ Un intégrateur de dépôt environnemental

☒ Un environnement de développement intégré

☐ Un éditeur de code source


 Un IDE c'est un environnement de développement intégré ou un logiciel de création d'applications qui contient entre autres un éditeur de code source.

**Question 4**

Il est possible de personnaliser ses balises dans Twig.

☒ Vrai

☐ Faux


 Cela fait partie de la flexibilité de Twig qui, grâce à son lexer et parser, permet une grande personnalisation.

**Question 5**

Il est nécessaire d'être sous Symfony pour utiliser Twig.

☐ Vrai

☒ Faux


 Vous pouvez par exemple installer Twig seulement pour votre projet PHP sans pour autant avoir à installer le framework Symfony.

## Exercice p. 13 Solution n°2

### Question 1

Quel symbole opérateur n'est pas un opérateur mathématique ?


- ☐ \*
- ☐ /
- ☒ ==
- ☐ %

 Ce symbole est un opérateur de comparaison.

### Question 2

Quel pourrait être le résultat de cette fonction Twig : `{{ random(17) }}` ?


- ☐ 20
- ☒ 1
- ☐ 16,33

 Cette fonction demande un résultat aléatoire d'un entier entre 0 et 17.

### Question 3

Une variable Twig est différente d'une variable PHP.


- ☒ Vrai
- ☐ Faux

 Twig appelle variable tous les éléments qui lui sont fournis à partir de PHP.

### Question 4

Si `foo = 'hello'` et `bar = 'world'`, quelle est la bonne concaténation pour obtenir « *hello world* » ?

- ☒ `{{ foo|upper ~ ' ' ~ bar ~ '!' }}`
- ☐ `{{ foo ~ bar |upper ~ '!' }}`
- ☐ `{{ 'foo ~ bar ~ '!' }}`

 Le symbole « ~ » sert à concaténer en Twig, le pipe « | » sert à ajouter un filtre, ici upper.

### Question 5

`{% extends "base.html.twig %}` est la bonne façon pour utiliser l'héritage dans une page html enfant.

- ☒ Vrai
- ☐ Faux

- Q Grâce à cette ligne de code insérée dans les pages enfants il est possible de modifier les blocs modèles de la page parent.

#### p. 14 Solution n°3

L'héritage de templates va me permettre de résoudre une problématique à savoir ne pas répéter sur chacun des templates la même chose. En fait, l'héritage, c'est un peu comme les `include()`, mais en bien mieux car il permet des adaptations plus subtiles. En créant une page modèle, que l'on appelle aussi page parent, cela permet de créer un squelette avec des éléments fixes qui seront toujours utilisés et des éléments qui seront modifiables et encapsulés par des balises « *Blocks* ». Il est même possible d'emboîter des blocs (le français de Blocks) dans des blocs. Ainsi, lorsque l'on créera des pages enfants, il suffira d'étendre la page parent, il n'y aura ainsi pas à recopier des éléments récurrents ou des imports communs à chaque page.

#### p. 14 Solution n°4

Une des fonctions particulièrement intéressantes de Twig, ce sont les filtres. Grâce à ceux-ci, au lieu de répéter à chaque fois une ligne de code par exemple, il est possible de créer un filtre qu'il suffira d'appeler à l'aide d'un pipe et du nom du filtre.

Il est vrai que cela nécessitera la création d'une classe extension qui étendra la classe `ExtractExtension` de Twig, mais en prenant le temps de bien paramétrer cette classe et d'y ajouter si nécessaire des conditions, cela permettra de ne pas avoir à le faire dans le fichier HTML et apportera de la clarté aux codes ainsi qu'un gain de temps une fois mis en place.

#### Exercice p. 14 Solution n°5

##### Question 1

Il est possible de faire un dump avec Twig.

- ☒ Vrai
- ☐ Faux
- Q Le débogueur dump sera appelé ainsi `{{ dump(mavARIABLE) }}`.


##### Question 2

Quelle variable n'est pas une variable globale native en Twig ?

- ☒ test
- ☐ user
- ☐ request
- Q test fait référence à des opérateurs de tests en Twig.


##### Question 3

Plus tôt, une variable « *montant* » a été déclarée. Quelle condition va permettre d'obtenir un résultat si la valeur de « *montant* » est supérieure à 20 ?

- ☒ `{% if montant > 20 %}`
- ☐ `{{% if montant > 20 %}}`
- ☐ `{% if montant < 20 %}`
-  La seconde proposition comporte deux accolades et la troisième proposition demande un chiffre strictement inférieur à 20.


#### Question 4

Si cette ligne de code est dans la page parent : `{% block header_img%} {% endblock %}`, il me suffira de l'appeler et d'ajouter mon image dans le bloc pour modifier l'image.

- ☒ Vrai
- ☐ Faux
-  Si un modèle a été créé, il suffit d'appeler les blocs dans la page enfant.

#### Question 5

Pour récupérer tous les éléments d'un tableau, je peux faire une boucle foreach en Twig.

- ☐ Vrai
- ☒ Faux
-  En Twig, il faut utiliser les expressions « *for* » et « *in* », sans oublier le « *endfor* » pour boucler un tableau.