Twig

Twig is a PHP template engine. It was created by Symfony developers. Twig files have the extension of `.html.twig`; they are a mix of static data such as HTML and Twig constructs.

Twig uses the double curly brace delimiters `{{ }}` for output and the curly brace percentage delimiters `{% %}` for logic. The `{# #}` are used for comments.

```
<ul>
    {% for word in words %}
        <li>{{ word }}</li>
    {% endfor %}
</ul>
```

This code is a sample Twig syntax. In this code, we use `for` tag to crate a loop.

Twig syntax consists of tags, filters, functions, operators, and tests.

Setting up Twig

First, we set up Twig.

```
$ composer require twig/twig
```

We install Twig with composer.

```
$ mkdir templates
```

We will place our template files into the `template` directory.

```
require __DIR__ . '/vendor/autoload.php';
```

We need to add the `autoload.php` file to our scripts.

Template engine

A template engine or template processor is a library designed to combine templates with a data model to produce documents. Template engines are often used to generate large amounts of emails, in source code preprocessing or producing dynamic HTML pages.

We create a template engine, where we define static parts and dynamic parts. The dynamic parts are later replaced with data. The rendering function later combines the templates with data.

Twig first example

The following is a simple demonstration of the Twig template system.

first.php

```php
<?php

require __DIR__ . '/vendor/autoload.php';

use Twig\Environment;
use Twig\Loader\FilesystemLoader;

$loader = new FilesystemLoader(__DIR__ . '/templates');
$twig = new Environment($loader);

echo $twig->render('first.html.twig', ['name' => 'John Doe',
    'occupation' => 'gardener']);
```

We used `FilesystemLoader` to load templates from the specified directory.

```php
echo $twig->render('first.html.twig', ['name' => 'John Doe',
    'occupation' => 'gardener']);
```

The output is generated with `render`. It takes two parameters: the template file and the data.

templates/first.html.twig

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <p>
        {{ name }} is a {{ occupation }}
    </p>

</body>
```

```
</html>
```

This is the template file. The variables are output with `{{}}` syntax.

```
$ php first.php
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <p>
        John Doe is a gardener
    </p>

</body>

</html>
```

This is the output.

Twig filters

Filters allow us to modify data in various ways.

filters.php

```
<?php

require __DIR__ . '/vendor/autoload.php';

use Twig\Environment;
use Twig\Loader\FilesystemLoader;

$loader = new FilesystemLoader(__DIR__ . '/templates');
$twig = new Environment($loader);

$words = ['sky', 'mountain', 'falcon', 'forest', 'rock', 'blue'];
$sentence = 'today is a windy day';

echo $twig->render('filters.html.twig',
    ['words' => $words, 'sentence' => $sentence]);
```

In the example, we have an array and a string as template data.

templates/filters.html.twig

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Filters</title>
</head>

<body>

    <p>
     The array has {{ words | length }} elements
    </p>

    <p>
     Joined array elements: {{ words | join(',') }}
    </p>

    <p>
     {{ sentence | title }}
    </p>

</body>

</html>
```

Filters are applied with the `|` character. The example counts words with `length`, joins array elements with `join` and modifies characters with `title`.

Twig custom filters

We can create custom filters with `Twig_Filter`.

customfilter.php

```
<?php

require __DIR__ . '/vendor/autoload.php';

use Twig\Environment;
use Twig\Loader\FilesystemLoader;

$loader = new FilesystemLoader(__DIR__ . '/templates');
$twig = new Environment($loader);
$twig->addFilter(new Twig_Filter('accFirst', 'accFirst'));

$sentence = 'šumivé víno';

echo $twig->render('customfilter.html.twig',
    ['sentence' => $sentence]);

function accFirst($value, $encoding = 'UTF8')
{
    $strlen = mb_strlen($value, $encoding);
    $firstChar = mb_substr($value, 0, 1, $encoding);
    $rest = mb_substr($value, 1, $strlen - 1, $encoding);

    return mb_strtoupper($firstChar, $encoding) . $rest;
}
```

We add a new filter called `accFirst`. It modifies only the first letter and also handles accents.

templates/customfilter.html.twig

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Custom filter</title>
</head>

<body>

    <p>
     {{ sentence | accFirst }}
    </p>

</body>

</html>
```

This is the template file, which uses the custom `accFirst` filter.

Twig loops

To create loops, we use the `for` tag.

looping.php

```
<?php

require __DIR__ . '/vendor/autoload.php';

use Twig\Environment;
use Twig\Loader\FilesystemLoader;

$loader = new FilesystemLoader(__DIR__ . '/templates');
$twig = new Environment($loader);

$words = ['sky', 'mountain', 'falcon', 'forest',
    'rock', 'blue', 'solid', 'book', 'tree'];

echo $twig->render('words.html.twig', ['words' => $words]);
```

We will loop an array of words.

templates/words.html.twig

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Words</title>
</head>

<body>

    <ul>
    {% for word in words %}
        <li>{{ word }}</li>
    {% endfor %}
    </ul>

    <ul>
    {% for word in words|slice(2, 4) %}
        <li>{{ word }}</li>
    {% endfor %}
    </ul>

</body>

</html>
```

In the template file, we loop over the `words` array and generate an HTML list. With `slice` filter, we can loop over a part of the array.

Twig looping with if & else

We can combine the `for` tag with the `if` tag and `else` tags.

looping2.php

```
<?php

require __DIR__ . '/vendor/autoload.php';

use Twig\Environment;
use Twig\Loader\FilesystemLoader;

$loader = new FilesystemLoader(__DIR__ . '/templates');
$twig = new Environment($loader);

$users = [
    ['name' => 'John Doe', 'active' => false],
    ['name' => 'Lucy Smith', 'active' => false],
    ['name' => 'Peter Holcombe', 'active' => false],
    ['name' => 'Barry Collins', 'active' => false]
];

echo $twig->render('activeusers.html.twig', ['users' => $users]);
```

We send an array of users to the template file.

templates/activeusers.html.twig

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

<p>Active users</p>

<ul>
    {% for user in users if user.active %}
        <li>{{ user.name }}</li>
    {% else %}
        <li>No users found</li>
    {% endfor %}
</ul>

</body>

</html>
```

We output the user's name of the `user.active` property is true. When there are no active users, the output from the `else` tag is shown.

Twig set tag

The `set` tag allows to set a value to a variable inside a template.

```
$words = ['sky', 'mountain', 'falcon', 'forest',
    'rock', 'blue', 'solid', 'book', 'tree'];

echo $twig->render('test.html.twig', ['words' => $words]);
```

We have a list of words.

```
{% set sorted = words | sort %}

<ul>
{% for word in sorted %}
    <li>{{ word }}</li>
{% endfor %}
</ul>
```

We sort the array with `sort` filter and assign the sorted array to the `sorted` variable with `set`.

Twig verbatim tag

The `verbatim` marks sections as being raw text that should not be parsed.

```
{% verbatim %}
    <ul>
    {% for word in words %}
        <li>{{ word }}</li>
    {% endfor %}
    </ul>
{% endverbatim %}
```

For instance, if we had a tutorial explain some Twig tag, we would need not to parse a piece of the demonstration.

Twig format filter

The `format` filter formats a given string by replacing the placeholders. It works like the `sprintf` function.

```
$name = "John Doe";
$age = 34;

echo $twig->render('formatfil.html.twig', ['name' => $name, 'age' => $age]);
```

We send two variables to the template.

```
{{ "%s is %d years old" | format(name, age) }}
```

We build the string with `format`.

Twig date function

The `date` function converts an argument to a date to allow date comparison.

```
$user = ['name' => 'John Doe', 'created_at' => '2011/11/10'];
```

```
echo $twig->render('datefun.html.twig', ['user' => $user]);
```

The user array has a `created_at` key.

```
{% if date(user.created_at) < date('-5years') %}
    <p>{{ user.name }} is a senior user</p>
{% endif %}
```

In the template, we compare two dates.

Twig automatic escaping

Twig automatically escapes certain characters such as < or >.

```
$twig = new Environment($loader, [
    'autoescape' => false
]);
```

Autoescaping can be turned off with the `autoescape` option.

```
$data = "<script src='http::/example.com/nastyscript.js'></script>";

echo $twig->render('autoescape.html.twig', ['data' => $data]);
```

Users could potentionally add dangerous input to the application. The inclusion of unknown JS file can be prevented with autoescaping.

```
<p>
The data is {{ data }}
</p>

<p>
The data is {{ data | raw }}
</p>
```

If autoescaping is enabled, we can show the raw input with the `raw` filter.

```
<p>
The data is &lt;script
src=&#039;http::/example.com/nastyscript.js&#039;&gt;&lt;/script&gt;
</p>

<p>
The data is <script src='http::/example.com/nastyscript.js'></script>
</p>
```

This partial output shows how the characters are escaped.

Twig tests

Twig tests allow to test data. The tests are applied with the `is` operator.

```
$words = ['', null, 'rock', '   ', 'forest'];
echo $twig->render('tests.html.twig', ['words' => $words]);
```

We have an array of words that contains empty, null, and blank elements.

```
<ul>
{% for word in words %}

    {% if word is null %}
    <p>null element</p>
    {% elseif word | trim is empty %}
    <p>Empty element</p>
    {% else %}
    <li>{{ word }}</li>
    {% endif %}

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

To deal with empty, blank, and null elements, Twig has `empty` and `null` tests.

Twig inheritance

Twig's template inheritance is a powerful feature which removes duplication and promotes maintenance.

inheritance.php

```
<?php

require __DIR__ . '/vendor/autoload.php';

use Twig\Environment;
use Twig\Loader\FilesystemLoader;

$loader = new FilesystemLoader(__DIR__ . '/templates');
$twig = new Environment($loader);

echo $twig->render('derived.html.twig');
```

This is the `inheritance.php` file. It renders the `derived.html.twig`, which extends from `base.html.twig`.

templates/base.html.twig

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}{% endblock %}</title>
</head>

<body>

{% block body %}{% endblock %}

</body>

</html>
```

The base layout defines two blocks which are replaced by children: `title` and `body`.

templates/derived.html.twig

```
{% extends 'base.html.twig' %}

{% block title %}Some title{% endblock %}

{% block body %}

The body contents

{% endblock %}
```

The derived child template inherits from the base template with the `extends` keyword. The two block define custom text.

```
$ php inheritance.php
<!DOCTYPE html><html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Some title</title>
</head>

<body>


The body contents


</body>

</html>
```

This is the output.

Symfony example

Twig is an integral part of Symfony framework. The next example shows steps to use Twig in a Symfony skeleton application.

```
$ composer create-project symfony/skeleton simple
$ cd simple
```

We create a new Symfony skeleton application and move to the project directory.

```
$ composer require maker annotations twig
```

We include some basic Symfony components including Twig.

```
$ php bin/console make:controller HomeController
```

We create a home controller.

src/Controller/HomeController.php

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class HomeController extends AbstractController
{
    /**
     * @Route("/home", name="home")
     */
    public function index()
    {
        $words = ['sky', 'blue', 'cloud', 'symfony', 'forest'];

        return $this->render('home/index.html.twig', [
            'words' => $words
        ]);
    }
}
```

In the home controller, we render the `index.html.twig` template and pass it the `$words` array to process.

templates/base.html.twig

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
    </head>
    <body>
        {% block body %}{% endblock %}
        {% block javascripts %}{% endblock %}
    </body>
</html>
```

This is the base layout page.

templates/home/index.html.twig

```
{% extends 'base.html.twig' %}

{% block title %}Home page{% endblock %}

{% block body %}

<h2>List of words</h2>

<ul>
{% for word in words  %}
    <li>{{ word }}</li>
{% endfor %}
</ul>

{% endblock %}
```

This is the home page template. It uses the `for` tag to iterate over the words and output them in an unordered list.

```
$ symfony server:start
```

We start the server.

We navigate to `http://localhost:8000/home` to see the result.