How Do We Use HTML Templates?

In this lesson, we will learn how to serve HTML templates in our Flask application.

WE'LL COVER THE FOLLOWING

Introduction

In the previous chapter, we learned how to create `URL` routes for our Flask application. Moreover, we also learned that we can return a `String` from the View function to be rendered on the front-end.

In this lesson, we will learn how to render **static templates**, instead of plain text, on the front-end using the Flask framework.

Static templates

Static templates are the `HTML` files which remain constant. In fact, the standard HTML files are static in nature. Whenever the same file is rendered, it shows the same output unless the file is updated.

Up until now, we have only been returning a `String` from the view function. But of course, in a real-world web application, we expect an `HTML` file to be

But of course, in a real-world web application, we expect an `HTML` file to be returned and rendered on the browser screen.

> **Note:** In the context of this course, we assume that you are already familiar with the basics of `HTML` and `CSS`.

Rendering HTML templates

`HTML` can be rendered in Flask using the following two methods:

A String

We can use the `HTML` as a `String` in the view function.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def home():
    return "<h1>Welcome to the HomePage!</h1>"


if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=3000)
```

`render_template()` function

Returning an `HTML` template as a `String` works perfectly—but it is not appropriate for practical applications. It is best to keep the code and the templates separate. Thus, we create separate files containing the `HTML` code for the templates. Then, these `HTML` files can be referred to inside the *views* by their names. Flask contains a function called `render_template()` which is used to render the desired HTML templates. This method has the following parameters:

- **`template_name_or_list`**: the name of a template or an iterable list of templates (the method will render the first template in the list)
- **`context`** *(optional)*: variables that should be available inside the template.

The output of the `render_template()` is then returned by the *view* instead of a simple `String`, as we were previously doing.

```
def view_name():
    return render_template(template_name)
```

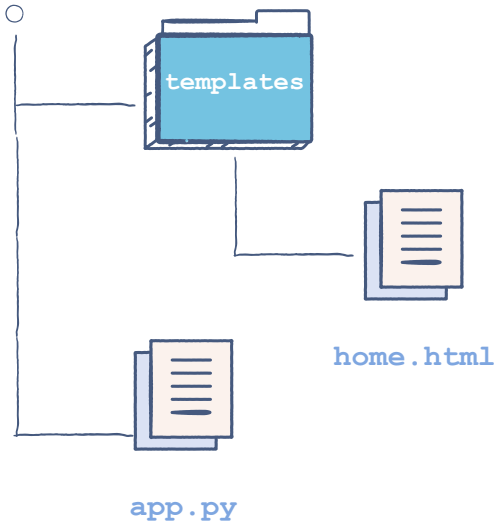
```
    return render_template(template_name)
```

File structure strategies

The Flask framework looks for the `HTML` template files in a directory named `\templates`. This directory can be placed per the following file structures so that Flask can find it.

Module file structure

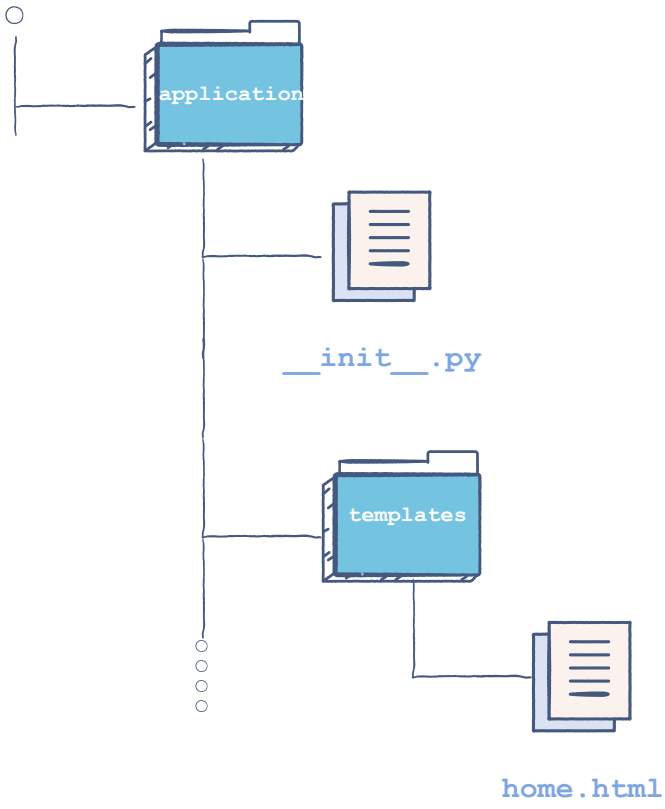
If we follow a modular file structure in our project (*i.e the application logic is in one `.py` file*), a separate directory called **"templates"** can be created in the same directory as the main module of the application *(i.e `app.py`)*.



Package File Structure

Package file structure

If the application logic is divided into separate modules *(i.e., separate `.py` files)*, then these files are present in the same package. As you may know, in Python, a package is simply a directory containing a file named `__init__.py`. If this structure is being followed in our application, then we will create the `templates` directory inside the main application package.



Module File Structure

Now, let's use the **package file structure** approach and render the `home.html` template by using `render_template()` function in the example above.

An example using `render_template()`

Let's take a look at the example application.

```
<!DOCTYPE html>
<html>
<h1>Welcome to the Homepage!</h1>
</html>
```

Explanation

In this application, we have created a `templates` directory containing an `HTML` file called `home.html`. Then, in the application module, `app.py`, we first import the `render_template` function in **line 1**. Then, the view function `home()` renders the `home.html` template by passing its name to `render_template()` and returning the output in **line 6**.

In the next lesson, we will again work on the **Paws Rescue Center** web application!

application?