COURSEWARE

Templating

Contents

Overview

In programming, there is a common phrase: **D**on't **R**epeat **Y**ourself (DRY).

When creating a web application, we don't want to have to write out an entire **HTML** page for each **route** we create.

In Flask, there is a templating library called **Jinja2** which allows you to avoid repeating HTML code.

Jinja2

Jinja2 is a templating library that allows us to refer to our Python code in our HTML files.

It also allows us to add logic to our HTML files, like iterative loops or conditional statements, which helps us avoid repeating ourselves.

We can also divide our templates using Jinja2 blocks.

Expressions

Double curly braces (`{{ }}`) are called **expressions** and are used for value substitution. Any variable or function invoked inside the braces will be substituted for its value, or set of values.

For example:

```
{{ my_variable }}
```

Will be substituted for whatever value has been assigned to the variable `my_variable`.

You can also specify expressions within the braces. For example:

```
{{ number * 2 }}
```

The value that will be substituted will the value of the variable `number` multiplied by 2.
The value that will be substituted will the value of the variable `number` multiplied by 2.

You can also invoke functions within substitution blocks. For example, we can substitute the URL for a route defined in our Flask application code into a hyperlink tag:

```
<a href="{{ url_for('index') }}">Home</a>
```

Statements

Curly braces with percentage symbols are called **statements** and are used to control the structure of a template.

```
{% statement %}
```

Blocks

Blocks allow us to split our code into different sections. We can refer to those sections so that our HTML is both modular but also not repetitive. A block is a piece of code to be executed so it needs to be surrounded but curly braces and percentage symbols.

```
{% block name_of_block %}
{% endblock name_of_block %}
```

Extending Files

We can extend templates to use its current structure.

```
{% extends 'example.html' %}
```

This is another Jinja2 property that allows us to avoid repeating code. By putting the piece of code above, everything from that file will be imported to our current template.

Unfortunately, this will override everything in your current file. For example:

`main.html`

```
<h1>HELLO</h1>
```

`secondary.html`

```
{% extends 'main.html' %}
<p>My name is Ben</p>
```

If we were to render `secondary.html` we would only see `HELLO`.

However, when being used in conjunction with blocks we can add new content to our files. For example:

`main.html`

```
<h1>HELLO</h1>
{% block body %}
{% endblock body %}
```

`ben.html`

```
{% extends 'main.html' %}
{% block body %}
<p>My name is ben</p>
{% endblock body %}
```

`harry.html`

```
{% extends 'main.html' %}
{% block body %}
<p>My name is harry</p>
{% endblock body %}
```

So here we create a block in the `main.html` and then refer to that block in `ben.html` and `harry.html`. Everything we place within the block will be rendered with the contents of the extended template.

So if we render `harry.html`, we would get.

```
HELLO My name is harry
```

But if we render `ben.html`, we would get.

```
HELLO My name is ben
```

`if` Statements

Jinja2 allows you to use conditionals with your webpages so that there can be a condition that has to be met before the relevant information appears on the page.

```
{% if 3 == 4 %}

<h1>3 is equal to 4</h1>

{% else %}

<h1>3 is not equal to 4</h1>

{% endif %}
```

Unless the laws of mathematics change, this template will always display:

```
3 is not equal to 4
```

If statements are code to be executed so they need to be surrounded by curly braces and percentage symbols.

Notice that there is an `{% endif %}`. This is needed to define the end of the statement.

`for` Loops

Repeating yourself when programming causes all sorts of maintainability issues and in some cases it's impossible to avoid using for-loops.

So imagine you have a page displaying all the users that have signed up for your website.

```
<p>Ben</p>
<p>Harry</p>
<p>Luke</p>
```

Instead of writing each user individually, you can write a for loop to iterate through them.

```
{% for user in ["Ben", "Luke", "Harry"] %}

<p>{{ user }}</p>

{% endfor %}
```

Rendering Templates

Now we have seen what a template can look like and contain, let's look at how to use them.

Flask has an in-built function `render_template()`, which takes a HTML filename then renders and returns the contents of the HTML file.

```
render_template('index.html')
```

> Note: In order to invoke the `render_template()` function, it must first be imported from the `flask` library:
>
> ```
> from flask import render_template
> ```

We can pass variables into the render_template function to be used inside the template.

```
render_template('index.html', users=['ben', 'luke', 'harry'])
```

Allowing us to rewrite our for loop as:

```
{% for user in users %}

<p>{{ user }}</p>

{% endfor %}
```

> Note: To reference a variable in any template you must pass the variable into the render_template function.

Templates Folder

The `render_template()` function will render your HTML and Jinja2 code for you.

The function looks for the files inside a `templates` folder. This folder needs to be in the same directory as the Python file that is invoking the `render_template()` function.

Tutorial

In this tutorial, we will use blocks and extends to create a layout.html and refer to layout.html to create the rest of our HTML templates.

Setup

This tutorial assumes you are working on an Ubuntu VM, at least version `18.04 LTS`.

First, install apt dependencies:

```
sudo apt update
sudo apt install python3 python3-venv python3-pip
```

Create a directory named `flask-templating` and make it your current working directory:

```
mkdir flask-templating && cd $_
```

We now need to create a Python virtual environment to install our pip requirements in. Create a new virtual environment named `venv` and activate it:

```
python3 -m venv venv
source venv/bin/activate
```

Finally, we need to install Flask with pip:

```
pip3 install flask
```

Project Structure

We need to create this file structure to complete this tutorial:

```
.
├── templates
│   ├── layout.html
│   ├── ben.html
│   └── harry.html
└── app.py
```

Run these commands to generate it:

```
mkdir templates
touch app.py templates/{layout,ben,harry}.html
```

Create the Application

Using a text editor of your choice, paste these blocks of code into their respective files:

1. `app.py`

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/ben')
def ben():
    return render_template('ben.html')

@app.route('/harry')
def harry():
    return render_template('harry.html')

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

2. `layout.html`

```
<html>
<body>
<h1>Hello</h1>
{% block body %}
{% endblock body %}
</body>
</html>
```

3. `ben.html`

```
{% extends "layout.html" %}
{% block body %}
<p>My name is Ben</p>
{% endblock body %}
```

4. `harry.html`

```
{% extends "layout.html" %}
{% block body %}
<p>My name is Harry</p>
{% endblock body %}
```

By doing so, we have created a Flask application with two routes: `/ben` and `/harry`.

Both will return the rendered templates of their respective filenames, either displaying 'My name is Ben' or 'My name is Harry'.

Start the Application

Run the application with the command:

```
python app.py
```

Access the Application

Now go to your application via your machine's public IP address on port `5000` and navigate to the `/ben` and `/harry` pages.

You should see that the 'Hello' header is there for both pages but Ben and Harry only appear on their respective pages.

Clean Up

To stop your Flask application running, navigate back to your terminal and press `Ctrl+C`. You should now have control over your terminal again.

To deactivate the virtual environment, run:

```
deactivate
```

If you wish to delete the virtual environment, run:

```
rm -rf venv
```

Exercises

Create a Flask application that passes this list of names to a template and within the template iterates through the list and only shows the names that contain the letter "b".

```
["ben", "harry", "bob", "jay", "matt", "bill"]
```