Flask Templates

1. Setting Up Flask and Jinja Templates

Creating a Flask Project: First, set up your Flask project. Ensure Flask is installed in your environment:

```
pip install Flask
```

Create a basic Flask application in a file named `app.py`:
python

```
from flask import Flask, render_template


app = Flask(__name__)


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


if __name__ == '__main__':
    app.run(debug=True)
```

-

Directory Structure: Flask looks for templates in a directory named `templates` by default. Your project should be organized like this:
arduino

```
/project_directory
```

```
├── app.py
└── templates
    └── home.html
```

-

2. Creating Jinja Templates

Basic HTML Template: Create an HTML file named `home.html` inside the `templates` directory:
html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home Page</title>
</head>
<body>
    <h1>Welcome to the Home Page!</h1>
    <p>This is a simple Flask application.</p>
</body>
</html>
```

-
- **Rendering the Template**: When the user visits the home route (`/`), the `home.html` template will be rendered by the `render_template` function.

3. Passing Dynamic Content to Templates

Passing Variables: You can pass dynamic content (variables) from your Flask view to the template using the `render_template` function.
python

```
@app.route('/')

def home():

    title = "Home Page"

    user = "Alice"

    return render_template('home.html', title=title,
user=user)
```

Update `home.html` to use the passed variables:
html

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>{{ title }}</title> <!-- Use the title variable -->

</head>

<body>

    <h1>Welcome, {{ user }}!</h1> <!-- Use the user variable
-->

    <p>This is a simple Flask application.</p>

</body>

</html>
```

-

- **Jinja2 Syntax**:
  - `{{ ... }}`: Used to evaluate expressions and print variables.
  - `{% ... %}`: Used for statements like loops, conditionals, etc.

Conditionals and Loops: Jinja2 allows you to use conditionals and loops within your templates.
python

```
@app.route('/')

def home():

    title = "Home Page"

    user = "Alice"

    items = ["Apples", "Bananas", "Cherries"]

    return render_template('home.html', title=title,
user=user, items=items)
```

Update `home.html` to display the list of items:
html

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>{{ title }}</title>

</head>

<body>

    <h1>Welcome, {{ user }}!</h1>

    <p>This is a simple Flask application.</p>
```

```
    <h2>Your Items:</h2>
    <ul>
        {% for item in items %}
            <li>{{ item }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

-

Using Conditionals:
html

```
{% if user %}
    <h1>Welcome, {{ user }}!</h1>
{% else %}
    <h1>Welcome, Guest!</h1>
{% endif %}
```

-

4. Extending Templates

Base Template: In larger applications, you might want to create a base template that other templates can extend. This allows you to avoid repeating common HTML structures (e.g., header, footer).
Create a base template (`base.html`):
html

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>{% block title %}My Flask App{% endblock %}</title>

</head>

<body>

    <header>

        <h1>My Flask App</h1>

    </header>


    <main>

        {% block content %}{% endblock %}

    </main>


    <footer>

        <p>&copy; 2024 My Flask App</p>

    </footer>

</body>

</html>
```

-

Extending the Base Template: Now, you can extend the `base.html` template in your `home.html`:

html

```
{% extends "base.html" %}


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


{% block content %}
    <h2>Welcome, {{ user }}!</h2>
    <p>This is a simple Flask application.</p>
{% endblock %}
```

-

5. Static Files

Serving Static Files: Flask automatically serves files from a directory named `static`. You can place CSS, JavaScript, images, and other static assets there.
Directory structure:
arduino

```
/project_directory
├── app.py
├── templates
│   ├── base.html
│   └── home.html
└── static
    └── style.css
```

-

Linking Static Files: Link to the static CSS file in `base.html`:
html

```
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
```

- The `url_for('static', filename='style.css')` generates the correct URL for the static file.

Summary

- **Jinja2 Templates**: Used to render dynamic HTML content in Flask.
- **Passing Variables**: You can pass variables from your Flask view to the template using `render_template`.
- **Jinja2 Syntax**: Use `{{ }}` for expressions and `{% %}` for control structures like loops and conditionals.
- **Template Inheritance**: Create reusable HTML structures using template inheritance with `{% extends %}` and `{% block %}`.
- **Static Files**: Serve CSS, JS, and images using the `static` folder.