INFO3180 Tutorial 2

Flask Routes, Static Files & Templates

In this tutorial we will review what we've learnt about Flask so far including Flask Routes and Templates.

Flask Review

What we know about Flask at this time:

- a Python Micro-framework for building web applications
- Flask uses specific conventions which help to save you configuring things, for example the **`static/`** folder and **`templates/`** folder are (by convention) the locations for static assets and templates respectively.
- Flask defines routes using decorators.

By now you would have deployed a simple application on Heroku that takes advantage of the following features of Flask

- routes
- templates
- static assets

A few important conventions of Flask

- Flask uses a library called **`Jinja2`** for templates.
- Templates by convention are stored in the **`"templates"`** directory.
- Static assets can be retrieved using the **`url_for()`** method and are usually stored in the **"static"** directory.

Flask app Starter Project

Moving forward with Flask it recommended that you treat your application like a module. This means that instead of a single **`app.py`** file you will now have an **`app`** folder. To distinguish it as a module it will contain an **`__init__.py`** file (see the diagram).

Sample application structure

When deploying Flask to Heroku the folder structure will also need to contain a **`Procfile`** and a **`requirements.txt`** file, the structure below does not show all the files but is a good summary of what your directory might look like when you're ready to deploy:

```
flask_starter/
├── app
│   ├── __init__.py
│   ├── static
│   ├── templates
│   ├── views.py
├── Procfile
├── README.md
├── requirements.txt
├── run.py
```

Flask Routes

The Flask documentation has a good overview of routes, you can read more here: http://flask.pocoo.org/docs/quickstart/#routing

Make sure you have a clear understanding of variable routes, things like:

```
@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post {}'.format(post_id)
```

A simple example

As a quick and simple example of using Flask routes we've provided the following repository:

https://github.com/uwi-info3180/flask_routes

POST vs GET

Sometimes you'll want a route to behave differently depending on the way that route is called. A route is mainly called either as a **POST** or a **GET** (though there are other types).

The simple example responds differently depending on if you use a **POST** or **GET** to send the request to certain routes. You can test this using the command line utility **`'curl'`**. If you prefer a graphical interface you can use the Postman or Insomnia desktop app instead of curl.

An example of how Flask differentiates between POST and GET requests can be seen in the following links:

https://github.com/uwi-info3180/flask_routes/blob/master/app/views.py#L44
https://github.com/uwi-info3180/flask_routes/blob/master/app/views.py#L55

Static Files

By convention you're encouraged to store your static files in a folder called **`'static'`**. You could keep your static files in other locations but you'd need to deliberately override the default settings.

A file called **`app.js`**, placed in the static folder would be accessible at the path **`'/static/app.js'`** and a file called **`app.css`** would be accessible at the path **`'/static/app.css'`**.

The snippet below shows how you would call them in a template:

```
<script src="/static/app.js"></script>
<link rel=stylesheet type=text/css href="/static/app.css">
```

You may also place these in sub-folders within the static folder and reference them accordingly. E.g. if you had your JavaScript code in a sub-folder called '**js**' and your css in sub-folder called '**css**', then you would reference them as **/static/js/app.js** and **/static/css/app.css.**

You could use the **url_for** method to reference your JavaScript and CSS files:

```
<script src="{{ url_for('static', filename="app.js") }}"></script>
<link rel=stylesheet type=text/css href="{{ url_for('static', filename="app.css") }}">
```

Serving Static files with routes

Static files can be served using the **app.send_static_file()** method. An example can be seen at the following link:

https://github.com/uwi-info3180/flask_routes/blob/master/app/views.py#L70

Here we can load a simple txt file by specifying the filename as part of the route.

Templates with Jinja2

Flask depends on a templating library called Jinja2. Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django's (another Python web framework) templates.

The syntax used in Jinja2 templates looks similar to this:

```
<title>{% block title %}{% endblock %}</title>
<ul>
{% for user in users %}
  <li><a href="{{ user.url }}">{{ user.username }}</a></li>
```

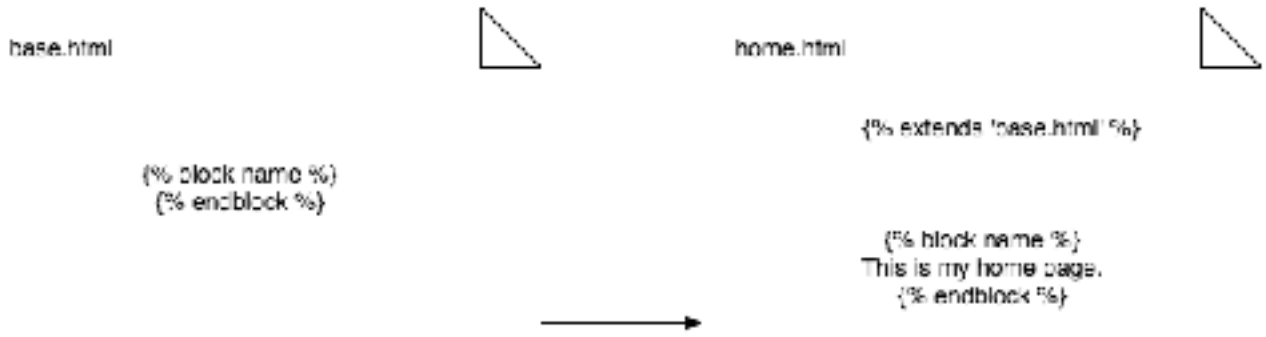
```
{% endfor %}
</ul>
```

The most important features of Jinja2 templating include:

- blocks and inheritance
- variables
- template logic

Blocks and Template Inheritance

The power of blocks is the ability to create parent templates which can be extended by child templates. The blocks act as replaceable areas which can be filled in by the child template.



e.g. Example of a block being defined in the base template **`base.html`**.
https://github.com/uwi-info3180/flask_starter/blob/master/app/templates/base.html#L28
The block name given in this case is 'main'.

Example of the **`base.html`** being extended by it's child template **`home.html`**
https://github.com/uwi-info3180/flask_starter/blob/master/app/templates/home.html

Variables

Variables are passed to Jinja2 templates from python code, to simplify this Flask provides a method called **`render_template()`**. Take a look at:

https://github.com/uwi-info3180/flask_starter/blob/master/app/views.py#L25

To output the variable you will use the syntax **{{ variable_name }}**.

Template Logic

Jinja2 templates also support simple logic such as **includes**, **for** loops and **if** statements. Template logic is defined using {% %}, for example:

```
{% if x < y %}
something will go here
{% endif %}
```

or

```
{% for user in users %}
something will go here
{% endfor %}
```

See for more details and examples: http://jinja.pocoo.org/docs/dev/templates/#list-of-control-structures