Flask 2: Templates & Jinja

Springboard

Download Demo Code

Goals

- Explain what HTML templates are, and why they are useful
- Use Jinja to create HTML templates for our Flask applications
- Debug our Flask applications more quickly by installing the Flask Debug Toolbar
- Serve static files (CSS, JS, etc) with Flask

Review

Views

Views are functions that return a **string** (a string of HTML)

Routes

Routes define the URL that will run a view function.

They are declared by using *decorators*.

A route and view function:

```
@app.route('/form')
def show_form():
    """Show greeting form."""

    return """
      <!DOCTYPE html>
      <html>
        <head>
          <title>Hi There!</title>
        </head>
        <body>
          <h1>Hi There!</h1>
          <form action="/greet">
            What's your name? <input name="person">
            <button>Go!</button>
          </form>
        </body>
      </html>
    """
```

This is kind of messy to read through (and we don't get nice things like color-highlighting in editors). Much better to keep HTML in a separate file.

Templates

How Can Templates Help?

- Produce HTML
- Allows your responses to be dynamically created
  - Can use variables passed from your views
  - For loops, if/else statements
- Can inherit from other templates to minimize repetition

Jinja

Jinja is a popular template system for Python, used by Flask.

There are many template systems for Python. Jinja is a particularly popular one. Django has its own template system, which served as an inspiration for Jinja.

Templates Directory

Your templates directory lives under your project directory. Flask knows to look for them there.

```
my-project-directory/
  venv/
  app.py
  templates/
    hello.html
```

Our Template

demo/templates/hello.html

```
<!DOCTYPE html>
<html>
<head>
  <title>This is the hello page</title>
</head>
<body>
  <h1>HELLO!</h1>
</body>
</html>
```

Rendering a Template

```
@app.route('/')
def index():
    """Return homepage."""

    return render_template("hello.html")
```

Will find ***hello.html*** in ***templates/*** automatically.

Flask Debug Toolbar

Ultra-useful add-on for Flask that shows, in browser, details about app.

Install add-on product:

```
(venv) $ pip3 install flask-debugtoolbar
```

Add the following to your Flask ***app.py***:

```
from flask import Flask, request, render_template
from flask_debugtoolbar import DebugToolbarExtension

app = Flask(__name__)
app.config['SECRET_KEY'] = "oh-so-secret"

debug = DebugToolbarExtension(app)

... # rest of file continues
```

Note: SECRET_KEY

For now, that secret key doesn't really have to be something secret (it's fine to check this file into your GitHub, and you can use any string for the SECRET_KEY.

Later, when we talk about security & deployment, we'll talk about when and how to actually keep this secret.

Using The Toolbar

Request Vars
: Explore what Flask got in request from browser

HTTP Headers
: Can be useful to explore all headers your browser sent

Templates
: What templates were used, and what was passed to them?

Route List
: What are all routes your app defines?

Dynamic Templates

Jinja will replace things like ***{{ msg }}*** with value of ***msg*** passed when rendering:

templates/lucky.html

```
<h1>Hi!</h1>

<p>Lucky number: {{ lucky_num }}</p>
```

app.py

```
@app.route("/lucky")
def show_lucky_num():
    "Example of simple dynamic template"

    num = randint(1, 100)

    return render_template("lucky.html",
                           lucky_num=num)
```

Example: Greeting

Let's make a form that gathers a user's name.

On form submission, it should use that name & compliment the user.

Our Form

demo/templates/form.html

```
<!DOCTYPE html>
<html>
<body>
  <h1>Hi There!</h1>
  <form action="/greet">
    <p>What's your name?  <input name="person"></p>
    <button>Go!</button>
  </form>
</body>
</html>
```

Our Template

demo/templates/compliment.html

```
<!DOCTYPE html>
<html>
<body>
  <p>Hi {{ name }}! You're so {{ compliment }}!</p>
</body>
</html>
```

Our Route

```
@app.route('/greet')
def offer_greeting():
    """Give player compliment."""

    player = request.args["person"]
    nice_thing = choice(COMPLIMENTS)

    return render_template("compliment.html",
                           name=player,
                           compliment=nice_thing)
```

Example 2: Better Greeting!

Let's improve this:

- We'll ask the user if they want compliments & only show if so
- We'll show a list of *3* random compliments, like this:

```
You're so:
<ul>
  <li>clever</li>
  <li>tenacious</li>
  <li>smart</li>
</ul>
```

Our Form

demo/templates/form-2.html

```
<!DOCTYPE html>
<html>
<body>
  <h1>Better Hi There!</h1>
  <form action="/greet-2">
    <p>What's your name? <input name="person"></p>
    <p>Want compliments?
      <input type="checkbox" name="wants_compliments">
    </p>
    <button>Go!</button>
  </form>
</body>
</html>
```

Our Route

```
@app.route('/greet-2')
def offer_better_greeting():
    """Give player optional compliments."""

    player = request.args["person"]

    # if they didn't tick box, `wants_compliments` won't be
    # in query args -- so let's use safe `.get()` method of
    # dict-like things
    wants = request.args.get("wants_compliments")

    nice_things = sample(COMPLIMENTS, 3) if wants else []

    return render_template("compliments.html",
                           compliments=nice_things,
                           name=player)
```

Conditionals in Jinja

`{% if CONDITION_EXPR %} ... {% endif %}`

```
{% if compliments %}
  You're so:
  ...
{% endif %}
```

Loops in Jinja

`{% for VAR in ITERABLE %} ... {% endfor %}`

```
<ul>
  {% for compliment in compliments %}
    <li>{{ compliment }}</li>
  {% endfor %}
</ul>
```

Our Template

demo/templates/compliments.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Hi {{ name }}!</h1>
{% if compliments %}
  <p>You're so:</p>
  <ul>
    {% for compliment in compliments %}
      <li>{{ compliment }}</li>
    {% endfor %}
  </ul>
{% endif %}
</body>
</html>
```

Template Inheritance

Motivation

Different pages on the same site are often 95% the same.

Repetition is Boring

Your templates have many things in common

```
<!DOCTYPE html>
<html>
<head>
  <title> TITLE GOES HERE </title>
  <link rel="stylesheet" href="/static/css/styles.css">
  <script src="http://unpkg.com/jquery"></script>
</head>
<body>
  <h1>Our Site</h1>
  BODY CONTENT GOES HERE
  <footer>Copyright by Whiskey.</footer>
</body>
</html>
```

If you want the same stylesheet everywhere, you have to remember to include it in every template. If you forget in one template, that page won't have your custom css that you spent so much time getting right. The same goes for scripts. If you want jquery everywhere, do you really want to have to remember to include it in the head in every template.

How to Use Template Inheritance

- Make a ***base.html*** that will hold all the repetitive stuff
- "Extend" that base template in your other pages
- Substitute blocks in your extended pages

Sample Base Template

`{% block BLOCKNAME %} ... {% endblock %}`

templates/base.html

```
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}TITLE GOES HERE{% endblock %}</title>
  <link rel="stylesheet" href="/static/css/styles.css">
  <script src="http://unpkg.com/jquery"></script>
</head>
<body>
  <h1>Our Site</h1>
  {% block content %}BODY CONTENT GOES HERE{% endblock %}
  <footer>Copyright by Whiskey.</footer>
</body>
</html>
```

Page Using Template

`{% block BLOCKNAME %} ... {% endblock %}`

templates/my-page.html

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

{% block title %}My awesome page title{% endblock %}

{% block content %}

  <h2>I'm a header!</h2>
  <p>I'm a paragraph!</p>

{% endblock %}
```

Where Other Project Files Go

Do I Need Routes for CSS (or JS, etc)?

```
@app.route("my-css.css")
def my_css():
    return """
      b { color: red }
      ...
    """
```

No! That would be tedious — plus, everyone gets the *same* CSS

Static Files: CSS and JS

In ***static/*** directory:

```
my-project-directory/
  venv/
  app.py
  templates/
    hello.html
  static/
    my-css.css
    my-script.js
```

Find files like:

```
<link rel="stylesheet" href="/static/my-css.css">
```

The static directory is where you put files that don't change, unlike templates, which are parsed. The static directory can be divided in to the types of files that it contains: js for javascript, css for css files, img for images, etc., but that isn't necessary.