CS1520 Recitation:

Flask 2: Templating

•••

Jeongmin Lee

Plan for Today

- Templating in Flask
- Jinja Tags
- Control Flow
- Static Files
- Template Inheritance

Templating

In previous example

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

@app.route('/')
def hello_world():
    return 'Hello World!'

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

- Everything to show would be written in a python code
- That is, what is seen **is coupled to** what is processed

Templating

- Let’s decouple this mechanism:
    - Place what is seen into a **template html** file
    - Place what is processed into a **python** file
- `render_template("_template.html")` links two components

```
# file structure:
./hello.py
./templates/_template.html
```

Templating

```
# _template.html

<!doctype html>
<html>
   <body>

      <h1>Hello {{ name }}!</h1>

   </body>
</html>
```

```
# hello.py

from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<user>')
def hello_name(user):
   return render_template('_template.html', name = user)

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

Run hello.py and visit

http://127.0.0.1/hello/yourname

Example code from tutorialspoint.com

Control Statement

(Recap) Jinja Tags

- `{{ ... }}`
  - Expression tag, contents are evaluated and place in the text
- `{% ... %}`
  - Statement tag, used to define Jinja constructs and issue flow control statements
- `{# ... #}`
  - Comment

Example: Expression tags

```
# _template.html

<!doctype html>
<html>
   <body>

      <h1>Hello {{ name }}!</h1>

   </body>
</html>
```

Example: Statement tags

```
<!doctype html>
<html>
   <body>

      {% if marks>50 %}
      <h1> Your result is pass!</h1>
      {% else %}
      <h1>Your result is fail</h1>
      {% endif %}

   </body>
</html>
```

Templating with if/else

```python
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<int:score>')
def hello_name(score):
   return render_template('hello.html',
                          marks = score)

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

```html
<!doctype html>
<html>
   <body>

      {% if marks>50 %}
      <h1> Your result is pass!</h1>
      {% else %}
      <h1>Your result is fail</h1>
      {% endif %}

   </body>
</html>
```

Loop in Template

- Python loop constructors can be used inside a template

-
  ```
  {% for x in a_list %}
        {{ x }}
  {% endfor %}
  ```

- Notice that inside loop,
  `{{ x }}` is used instead of `{% x %}`

Loop in Template with Dictionary

```python
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/result')
def result():
   dict = {'phy':50,'che':60,'maths':70}
   return render_template('result.html',
                          result = dict)

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

```html
<!doctype html>
<html>
   <body>

      <table border = 1>
         {% for key, value in result.iteritems() %}

            <tr>
               <th> {{ key }} </th>
               <td> {{ value }} </td>
            </tr>

         {% endfor %}
      </table>

   </body>
</html>
```

Static Files

Static files

- There are some files that are not dynamically changed its contents while being used.
- Example: CSS and Javascript files

Static files

- There are some files that are not dynamically changed its contents while being used.
- Example: CSS and Javascript files
- While templating, you would be confused about routings (locating location of a file: relative path?)
- In Jinja, there is a solution.

Static files

- Place all of your static files (.css, .js) in ./static folder.
- In template html file, source files with
  `{{ url_for('static', filename = 'hello.js') }}`

```
<html>
   <head>
      <script type = "text/javascript"
         src = "{{ url_for('static', filename = 'hello.js') }}" ></script>
   </head>

   <body>
      <input type = "button" onclick = "sayHello()" value = "Say Hello" />
   </body>
</html>
```

Template Inheritance

•••

Template Inheritance

- Template inheritance allows you to build a base "skeleton" template that contains all the **common elements of your site** and defines **blocks** that child templates can override.

Base Template

- base.html defines a simple HTML skeleton document that you might use for a simple two-column page.
- The `{% block %}` tags define four blocks that child templates can fill in.
- All the block tag does is tell the template engine that a child template may override those placeholders in the template.

base.html : provides skeleton of a website

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
    {% endblock %}
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
        &copy; Copyright 2008 by <a
href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

Child Template

- Child.html : template for specific parts of the website
- You must have `{% extends %}` tag at the beginning.
  - It tells the template engine that this template "extends" another template (e.g., base.html).
  - When the template system evaluates this template, it first locates the parent.

child.html : template for a specific part of the website

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>
    <p class="important">
      Welcome to my awesome homepage.
    </p>
{% endblock %}
```

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Web
    {% endblock %}
</head>
<body>
    <div id="content">
        {% block content %}{% endblock %}
    </div>
    <div id="footer">
        {% block footer %}
        &copy; Copyright 2008 by <a
href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

child.html

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>
    <p class="important">
      Welcome to my awesome homepage.
    </p>
{% endblock %}
```

- Note that since the child template doesn’t define the footer block, the value from the parent template is used instead.
  You can access templates in subdirectories with a slash:
  {% extends "layout/default.html" %}

Super Blocks

- It’s possible to render the contents of the parent block by calling super.
- This gives back the results of the parent block.

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>
```

Questions?