Flask, Part 2

CST 205

Review

```python
# day2.py

from flask import Flask, render_template
from flask_bootstrap import Bootstrap

app = Flask(__name__)
bootstrap = Bootstrap(app)

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

Review (cont.)

```
 <!-- home.html -->

{% extends 'bootstrap/base.html' %}
{% block title %}
  Day 2
{% endblock %}

{% block content %}

  <div class="container">
    <h1>Day 2</h1>
  </div>


{% endblock %}
```

Jinja2

- *Jinja2* is a popular templating engine for Python.
- Combines a template with certain data source to render dynamic web pages.

Template inheritance

- As seen in our Bootstrap example, we can **extend** a base template:
  - `{% extends 'bootstrap/base.html' %}`
    - This must be the first tag.
- The format for a content block is:
  - `{% block content %} ... {% endblock %}`

Jinja2 template arguments

- We have seen `render_template()` used with one argument, the template filename.
    - We can pass in additional template arguments, e.g.

      ```
      return render_template('home.html', var1='1', var2='2')
      ```

    - We can refer to these template arguments in our template by surrounding them with `{{ }}`

Code sample

```
{% block content %}

  <div class="container">

    <h1>Day {{var2}}</h1>

    <p>{{var1}} is less than {{var2}}</p>
  </div>

{% endblock %}
```

Template arguments

- Additional template arguments can contain more complicated data, such as lists and dictionaries

- Suppose we have the following dictionary defined:

```
my_info = {
    'days': ['sun', 'mon', 'tues'],
    'flavors': ['sweet', 'sour'],
    'colors': ['blue', 'green', 'brown']
}
```

- We can pass this dictionary to a template:

```
@app.route('/')
def home():
    return render_template('home.html', s_list=my_info)
```

Loop in HTML

We passed a dictionary to our template and assigned it the variable `s_list`.

Here is how we loop through part of this dictionary:

```
<!-- templates/home.html -->

<ul>
  {% for item in s_list['days'] %}
    <li> {{ item }} </li>
  {% endfor %}
</ul>
```

Links

- HTML stands for Hypertext Markup Language.
  - **Hypertext** is another term for links
    - Traditional HTML uses anchor tags with `href` attribute for links, for example:

```
<a href="https://csumb.edu/">CSUMB</a>
```

Flask links

- We can use any standard HTML with Flask, but Flask also provides another approach with `url_for(view)`
    - `url_for()` generates a URL to the given view and allows us to pass arguments to the view.

Example

```python
# day2.py
# new view function
@app.route('/page2')
def page2func():
    return render_template('page2.html')
```

```html
<!-- templates/home.html -->
<div class="container">
  <h1>Day 2</h1>
  <a href="{{url_for('page2func')}}">Click me!</a>
</div>
```

```html
<!-- templates/page2.html -->
<div class="container">
  <h2>Welcome to page 2</h2>
</div>
```

Example with additional arguments and `if-else`

```html
<!-- templates/home.html -->
<div class="well well-lg">
  <a href="{{ url_for('page2func', mood='happy') }}">Happy</a>
  <br><br>
  <a href="{{ url_for('page2func', mood='sad') }}">Sad</a>
</div>
```

```python
# day2.py
@app.route('/page2/<mood>')
def page2func(mood):
    return render_template('page2.html', your_mood=mood)
```

```html
<!-- templates/page2.html -->
{% if your_mood=="happy" %}
  <p>Glad to hear things are going well!</p>
{% else %}
  <p>Turn that frown upside down!</p>
{% endif %}
```

Forms

- Web applications should be able to get information from the user, process it, and respond.
- Forms are easy to set up using HTML's `<form>` tag, but are very tricky to get right (see PRG, CSRF, etc.)
- WTForms is a popular Python package which provides sophisticated form handling.
    - Flask-WTF is a version of WTForms designed to easily integrate with Flask.

Flask-WTF Installation

- Install Flask-WTF (with virtual environment activated):

```
pip install Flask-WTF
```

- Full example here

Post-Redirect-Get

- We view the page containing the form (using a `GET` request)
- The form is submitted (using a `POST` request)
- If there are no errors, we use `redirect()` to go to another page.

Message flashing

- Record messages at the end of a request
    - Use `flash()` to send the messages
- Access the messages at the next request (and only then)
    - Use `get_flashed_messages()` to retrieve the messages