INFO3180 - LECTURE 2

FLASK

https://www.flickr.com/photos/canon1777/6154700518/

What is a framework?

A (web) framework is a collection of packages which helps developers to build websites, web applications and web services easier and faster.

SOME ADVANTAGES OF USING A FRAMEWORK

- Don't re-invent the wheel. Frameworks have common functionality for web applications already built. e.g. routing, templating, user authentication, database access, etc.
- Speed of development. Since the common tasks are already taken care of, you can use these to help you to focus on building the new and unique parts of your application faster.
- Security - Many frameworks can also help you to avoid some basic and common security mistakes such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking.

SOME POPULAR SERVER-SIDE WEB FRAMEWORKS

- PHP
  - Laravel
  - Symfony
  - Zend Framework
- Ruby
  - Ruby on Rails
- Python
  - Django
  - Flask

And there are many others for these and other languages.

FLASK

Flask is a micro-framework for Python.

http://flask.pocoo.org

A micro-framework like Flask, gives us just enough to be able to build a simple, yet powerful web application or API. Anything else can be added with other libraries (or packages) as needed.

FEATURES

- built-in development server and debugger
- integrated unit testing support
- RESTful request dispatching
- uses Jinja2 for templating
- support for secure cookies (client side sessions)
- 100% WSGI compliant

FEATURES

- Unicode based
- great documentation
- free and open source

URL RESOLUTION/ ROUTING

```
http://example.com/some-folder/
mypage.html
```

```
http://example.com/about/
person.php?name=Mary
```

mypage.html **would then be a file that is stored in a folder called** *some-folder* **in your web server document root. e.g.** */var/www/some-folder/mypage.html*

Likewise *person.php* would then be a file that is stored in a folder called *about* in your web server document root. e.g. */var/www/about/person.php*

But what if we change the name of the file? Also could make these URL's more beautiful and easy to remember?

```
http://example.com/about/mary
```

Here we don't need to have a file extension. And it doesn't even have to be the same name as the actual folder/file that contains our code.

This is where *routing* comes in.

You may also hear people mention *route mapping* or *URL dispatching*.

In Flask we can specify our routes and map them to functions. This URL resolution is achieved by a mechanism called url route registration.

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

@app.route("/")                              ⟶   http://example.com:5000/
def hello():
    return "Hello World"

@app.route("/profiles")                      ⟶   http://example.com:5000/profiles
def profiles():
    return "List of profiles"

if __name__ == "__main__":
    app.run(host=0.0.0.0)
```

You can also have placeholders to accept variables

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

@app.route("/profile/<username>")
def profile(username):
    return "User {0}".format(username)

if __name__ == "__main__":
    app.run(host=0.0.0.0)
```

$\longrightarrow$

```
http://example.com:5000/profile/
marsha
```

TEMPLATING

Templates contain the HTML served by your application and help to separate your application logic from your presentation logic.

Flask uses a templating engine called *Jinja2* and these templates are usually stored in a *templates* directory

Templates will contain *variables* and/or *expressions*, which get replaced with values when a template is *rendered*; and *tags*, which control the logic of the template.

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

@app.route("/profile/<username>")
def profile(username):
    return render_template("hello.html", name=username)

if __name__ == "__main__":
    app.run(host=0.0.0.0)
```

Here the `render_template()` method is used to tell Flask to load `hello.html` template for the profile method.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>My Heading</h1>
    <p>Hello {{ name }}</p>
    <p>Your To do list is:</p>
    <ul>
      {% for item in items %}
        <li>{{ item.caption }}</li>
      {% endfor %}
    </ul>
  </body>
</html>
```

TEMPLATE DELIMITERS

There are a few kinds of delimiters. The default Jinja2 delimiters are configured as follows:

- {% ... %} for Statements e.g. if, for, while, etc.
- {{ ... }} for Expressions to print to the template output
- {# ... #} for Comments not included in the template output

TEMPLATE INHERITANCE

- One of the most powerful parts of Jinja is ***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.

TEMPLATE INHERITANCE

base.html (Base/Parent Template)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    {% block main %}
    {% endblock %}
  </body>
</html>
```

hello.html (Child Template)

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

{% block main %}
<h1>My Heading</h1>
<p>Hello {{ name }}</p>
<p>Some other text.</p>
{% endblock %}
```

STATIC RESOURCES

Static resources include things like your *JavaScript*, *CSS*, *Images* and *File Uploads*.

Flask stores files like these in a directory called *static*.

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

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

RESOURCES

- Flask Documentation
- Flask Quickstart - http://flask.pocoo.org/docs/0.12/quickstart
- Jinja2 documentation - http://jinja.pocoo.org/docs/2.9/templates/

DEMO