Flask Intro

Springboard

Download Demo Code

Goals

- Describe the purpose and responsibilities of a web framework
- Build small web applications using Python and Flask
- Set environmental variables for local Flask development
- Handle GET and POST requests with Flask
- Extract data from different parts of the URL with Flask

Web Frameworks

A Quick Demo

```
(venv) $ FLASK_ENV=development flask run
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 166-886-783
```

What is a Web Server?

A program that's running on a machine and waiting for a web request.

Note: A web server is a technology that can process requests and issue responses via HTTP, a protocol used to distribute information on the world wide web. Though it's also used to refer to computer systems and even internet "appliances," we'll use web server to refer to the software running on a machine that's waiting to respond to HTTP requests.

The server then responds with the exact HTML text for that page:

Web Application

The ability to start a server in "listening" for requests, and then issue responses:

```
GET /hello     (http://server/hello)
```

↓

```
<html>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

Note: To keep code samples short in the presentation, we're eliding some less-important HTML markup. The shortest valid HTML skeleton in modern HTML would actually be:

```
<!doctype html>
<html>
  <head>
    <title>Hello!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

Flask is a Web Framework

- Set of functions, classes, etc. that help you define
  - **Which** requests to respond to
    - **http://server/about-us**
    - **http://server/post/1**
  - **How** to respond to requests
    - Shows an "About Us" page
    - Show the 1st blog post
- Like a library, but bigger and more opinionated
- Usage is similar to the Python Standard Library.

Using the Python Standard Library

```
from random import choice, randint
```

Using Flask

```
from flask import Flask, request
```

What Do Web Applications Need To Do?

- handle web requests
- produce dynamic HTML
- handle forms
- handle cookies
- connect to databases
- provide user log-in/log-out
- cache pages for performance
- & more!

Flask Apps

Installing Flask

```
$ python3 -m venv venv
$ source venv/bin/activate

(venv)$ pip3 install flask
  ... lots of stuff
Successfully installed flask Werkzeug Jinja2 ...
Cleaning up...
```

Making An App

Need to create a "flask application":

```
from flask import Flask

app = Flask(__name__)
```

When we create a Flask application, it needs to know what module to scan for things like routes (covered later)–so the `__name__` is required and should always be written like that.

Running Flask App

```
(venv) $ flask run
```

(Control-C to quit)

If your Flask app file isn't called **app**:

```
(venv) $ FLASK_APP=app.py flask run
```

`FLASK_APP=app.py` is passing an "environmental variable".

Only has this meaning while this program is running

Development Mode

Better to run Flask in "development mode":

- Much better error messages
- Automatically re-loads server when code changes on disk

Both of these are very helpful when developing– and very bad for working on a live, production server.

```
(venv) $ FLASK_ENV=development flask run
```

Setting Environmental Variables

Can set *FLASK_DEV* once per terminal session:

```
(venv) $ export FLASK_ENV=development
```

Add that line to shell config to run on every new terminal session.

Adding Routes

Making Responses

- A function that returns web response is called a **view**
  - Response is a **string**
  - Usually, a **string** of HTML
- So, our function returns an HTML string:

```
@app.route('/hello')
def say_hello():
    """Return a hello "Hello" greeting."""

    html = "<html><body><h1>Hello</h1></body></html>"
    return html
```

Handling Requests

On requesting **http://localhost:5000/hello** in browser, function is called:

```
@app.route('/hello')
def say_hello():
    """Return a simple "Hello" greeting."""

    html = "<html><body><h1>Hello</h1></body></html>"
    return html
```

- Flask lets you "route" a URL to a function
- `@app.route('/hello')` is a Python "decorator"
  - **"/hello"** in the decorator maps directly to the URL the user requested

Now we can get to this at http://localhost:5000/hello

Serving at the Root

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

    return """
        <html>
          <body>
            <h1> we the landing page</h1>
          </body>
        </html>
        """
```

This function will get called if the user requests **http://localhost:5000/**.

Now we can reach this page at http://localhost:5000

What Routes Return

Routes should return *strings*!

GET and POST

Requests

Flask provides an object, **request**, to represent web requests

```
from flask import request
```

Handling Query Arguments

For a url like **/search?term=fun**

```
@app.route("/search")
def search():
    """Handle GET requests like /search?term=fun"""

    term = request.args["term"]
    return f"<h1>Searching for {term}</h1>"
```

request.args is a dict-like object of query parameters.

Handling POST Requests

By default, a route only responds to GET requests

To accept POST requests, must specify that:

```
@app.route("/my/route", methods=["POST"])
def handle_post_to_my_route():
    """..."""
```

Example POST Request

```
@app.route("/add-comment")
def add_comment_form():
    """Show form for adding a comment."""

    return """
        <form method="POST">
          <input name="comment">
          <button>Submit</button>
        </form>
        """
```

```
@app.route("/add-comment", methods=["POST"])
def add_comment():
    """Handle adding comment."""

    comment = request.form["comment"]

    # TODO: save that into a database!

    return f"<h1>Received '{comment}'.</h1>"
```

request.form is a dict-like object of POST parameters.

Variables in a URL

Motivation

- Want user info pages for each user:
  - http://localhost:5000/user/whiskey
  - http://localhost:5000/user/spike
  - We don't want every possible username as a separate route
- Want to show blog posts (read from database) by id:
  - http://localhost:5000/post/1
  - http://localhost:5000/post/2

Variables in a URL

Argument capture in Flask:

```
USERS = {
    "whiskey": "Whiskey The Dog",
    "spike": "Spike The Porcupine",
}

@app.route('/user/<username>')
def show_user_profile(username):
    """Show user profile for user."""

    name = USERS[username]
    return f"<h1>Profile for {name}</h1>"
```

- ***<variable_name>* in *@app.route***
- View function must have same ***var_name*** as parameter

Can also specify *int* variable:

```
POSTS = {
  1: "Flask is pretty cool",
  2: "Python is neat-o"
}

@app.route('/post/<int:post_id>')
def show_post(post_id):
    """Show post with the given integer id."""

    print("post_id is a ", type(post_id))

    post = POSTS[post_id]

    return f"<h1>Post #{post_id}</h1><p>{post}</p>"
```

- ***<int:variable_name>* in *@app.route***
- Converts to integer when calling function

Can have more than one:

```
@app.route("/products/<category>/<int:product_id>")
def product_detail(category, product_id):
    """Show detail page for product."""

    ...
```

Query Params vs URL Params

http://toys.com/shop/spinning-top?color=red

```
@app.route("/shop/<toy>")
def toy_detail(toy):
    """Show detail about a toy."""

    # Get color from req.args, falling back to None
    color = request.args.get("color")

    return f"<h1>{toy}</h1>Color: {color}"
```

Which Should I Use?

URL Parameter	Query Parameter
/shop/<toy>	**/shop?toy=elmo**
Feels more like "subject of page"	Feels more like "extra info about page"
	Often used when coming from form

Looking Ahead

Coming Up

- HTML templates
- Handling cookies
- APIs and Flask
- Using databases with Flask
- Auto-generating forms
- Handling users and log in

Flask Documentation

- The Flask documentation (http://flask.pocoo.org/)