Flask Web Apps

Miles Erickson

August 18, 2017

Lecture Credits

- Emily Spahn
- Tetyana Kutsenko

Objectives

At the end of the morning, you'll be able to:

- Describe example data product workflows
- Implement simple webpages using HTML and Flask
- Describe the HTTP methods GET and POST & list the differences

Data Products & Workflow

- What do you deliver as a data scientist? (Think about case studies)

Data Products & Workflow

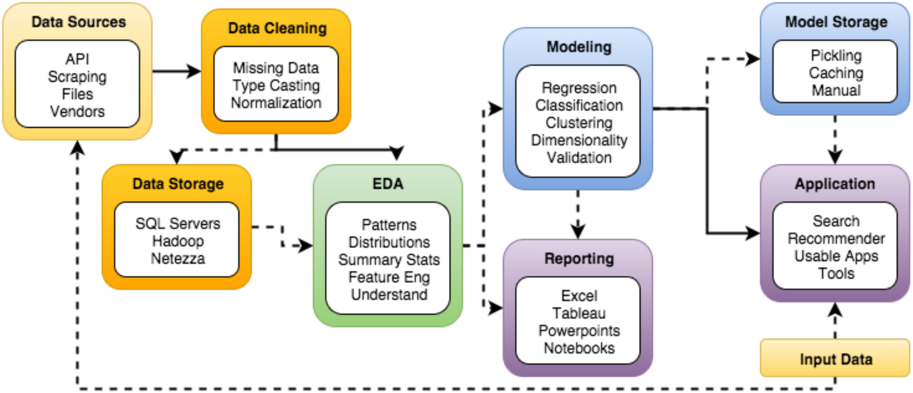


Figure 1:Data Products Flowchart

Image by Ming Huang

Why learn how to build web application

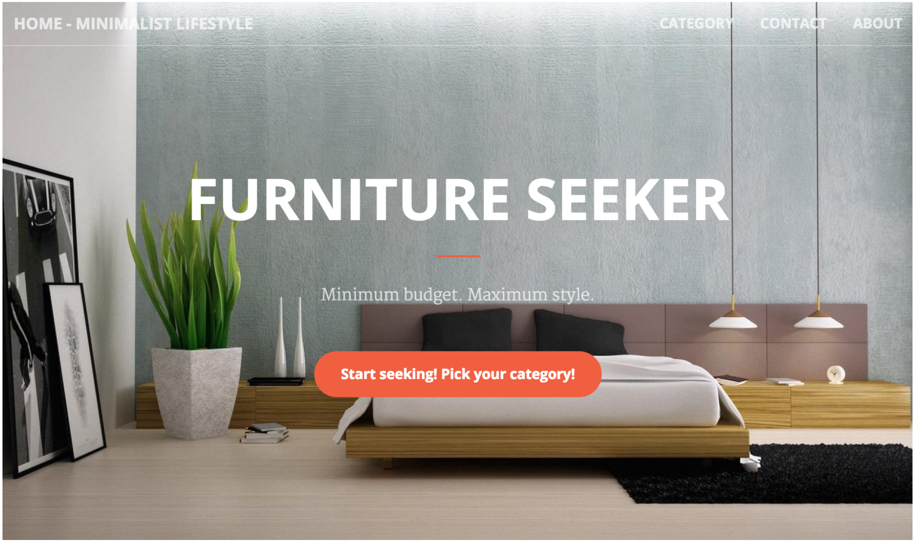

HOME - MINIMALIST LIFESTYLE
CATEGORY
CONTACT
ABOUT
FURNITURE SEEKER
Minimum budget. Maximum style.
Start seeking! Pick your category!

Figure 2:Furniture seeker

Why learn how to build web application

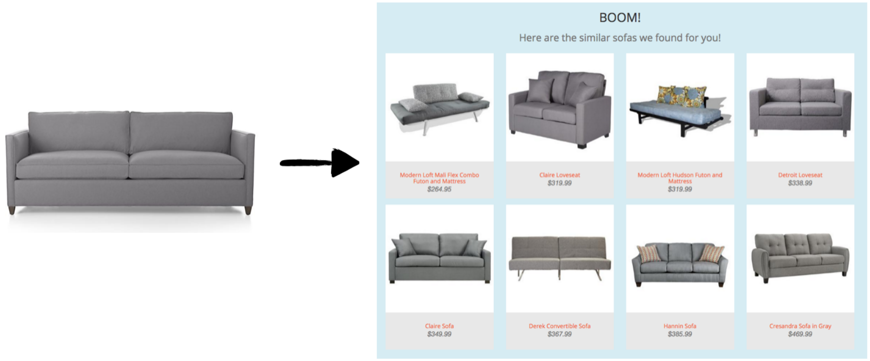
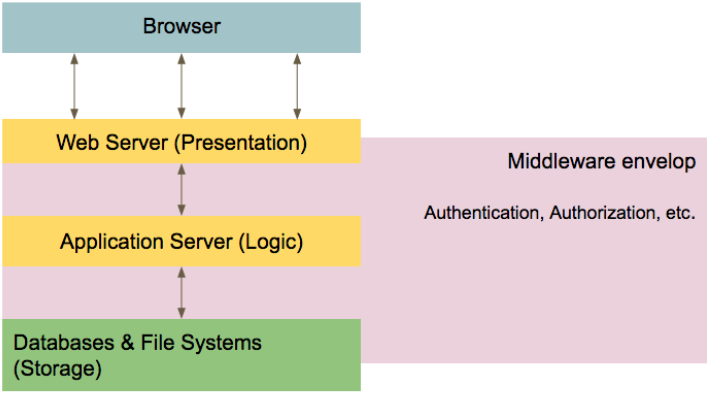


Figure 3:Furniture seeker

Galvanize Data Science Program - Capstone Project - Lili Yao

Web application

- Web application is a client–server software which is run in a web browser.
- Developing web application is simplified by frameworks such as Ruby on Rails, Django (Python), or Ember.js.


Browser
Web Server (Presentation)
Middleware envelop
Authentication, Authorization, etc.
Application Server (Logic)
Databases & File Systems (Storage)

HTTP methods: GET and POST

The Hypertext Transfer Protocol (HTTP) enables communications between clients and servers.

The two most common HTTP methods are:

- **GET**: Requests data from a server. (Default method in http & flask)
- **POST**: Submits data to server

Other HTTP Request Methods:

- PUT - Updates data on server
- DELETE - Deletes data on server

Important differences: see table at this w3 link

Review: HTML & CSS

Need basic HTML to build websites

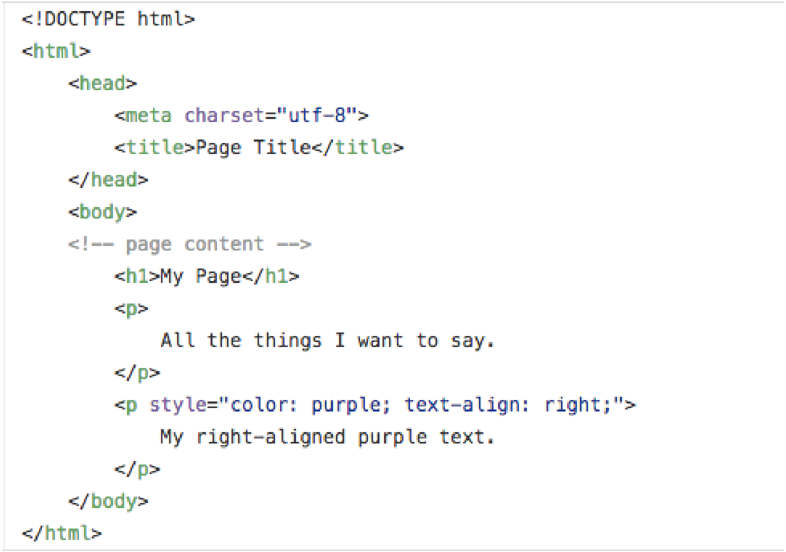
- HTML (Hyper Text Markup Language)
    - Based on markup tags
    - Each tag describes different document entity
- CSS (Cascading Style Sheets)
    - Describes how HTML is displayed on screen
    - Assigns style properties to (sections of) your site
    - Can control the layout of multiple web pages all at once

Review: HTML & CSS

Your main reference today is W3 Schools. They cover:

- HTML
- CSS
- JavaScript
- Bootstrap

Example



```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Page Title</title>
    </head>
    <body>
    <!-- page content -->
        <h1>My Page</h1>
        <p>
            All the things I want to say.
        </p>
        <p style="color: purple; text-align: right;">
            My right-aligned purple text.
        </p>
    </body>
</html>
```

Figure 4:HTML example

Flask

A Flask is a microframework for Python. “Micro” **does not mean**:

- Your whole web application has to fit into a single Python file (although it certainly can)
- Flask is lacking in functionality

It means:

- Flask aims to keep the core simple
- Flask won’t make many decisions for you
- Decisions that it does make are easy to change

Installation

- **Install** using ‘pip install flask’

Jinja2 is a templating language for Python

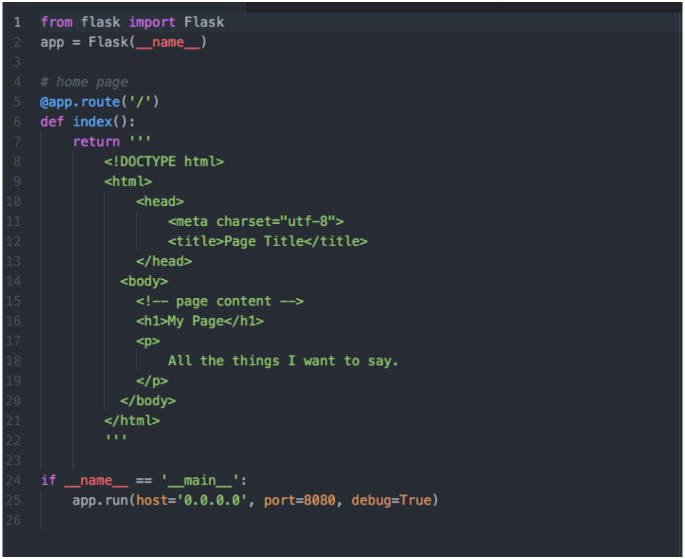
Flask Conventions

By convention

- templates - subdirectory for html template files
- static - subdirectory for files like css, js, font, images

Organize your files for flask (Reference)

Simple Flask application



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

# home page
@app.route('/')
def index():
    return '''
        <!DOCTYPE html>
        <html>
            <head>
                <meta charset="utf-8">
                <title>Page Title</title>
            </head>
          <body>
            <!-- page content -->
            <h1>My Page</h1>
            <p>
                All the things I want to say.
            </p>
          </body>
        </html>
        '''

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)

```

Figure 5:Flask application

Simple Flask application

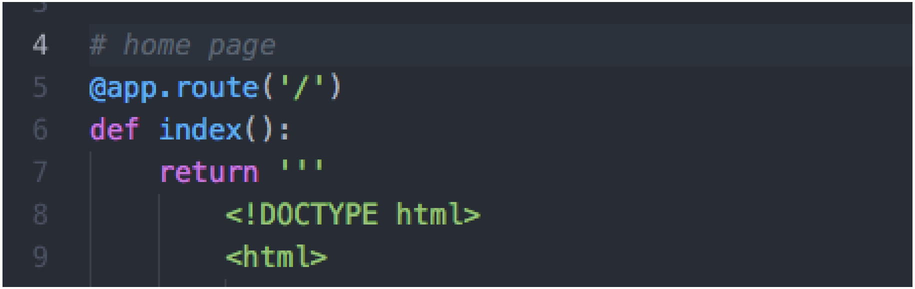
Run ‘python example.py’

Open in browser ‘http://localhost:8080/’ or ‘http://0.0.0.0:8080/’

Routing

Routing is binding URLs to Python functions.

The route() call is used to bind a function to a URL.

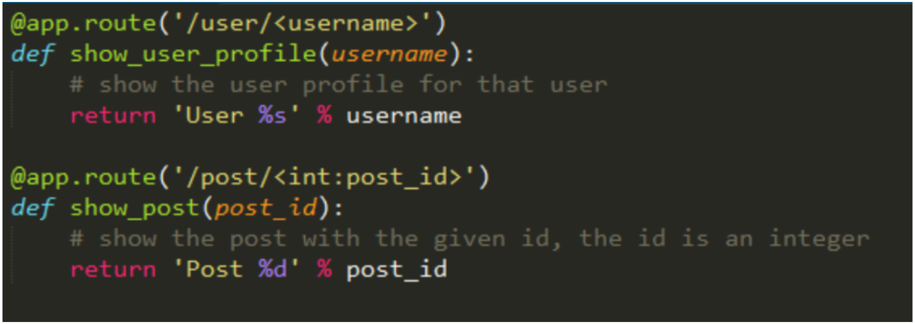


```
# home page
@app.route('/')
def index():
    return '''
        <!DOCTYPE html>
        <html>
```

Figure 6:Route

Routing

URL can contain variables (they are passed to bound function).



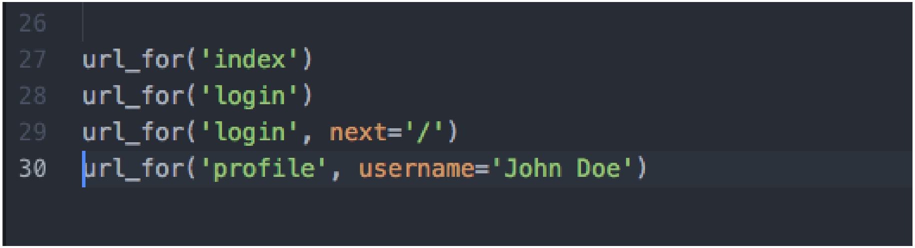
```python
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id
```

Figure 7:Route variable

Routing

You can generate URL for route with url_for() function.



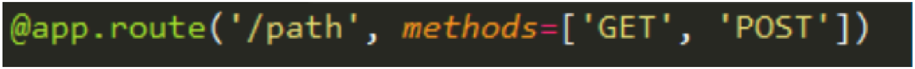
```

url_for('index')
url_for('login')
url_for('login', next='/')
url_for('profile', username='John Doe')
```

Figure 8:Route urls

Routing

By default, a route only answers to GET requests, but you can add the 'methods' argument to the route() call.

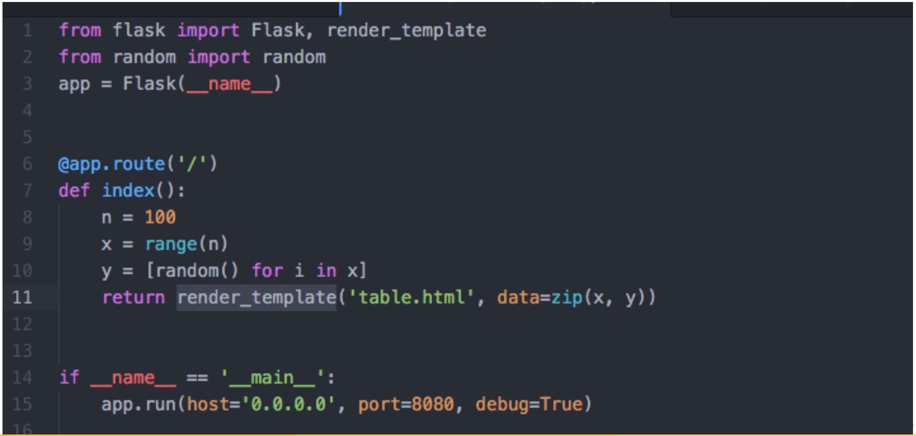


```
@app.route('/path', methods=['GET', 'POST'])
```

Figure 9:Route methods

Templates

- Generating HTML from within Python is not fun
- Template engine provides handy language to describe dynamic HTML
- Use render_template() from Jinja2 template engine

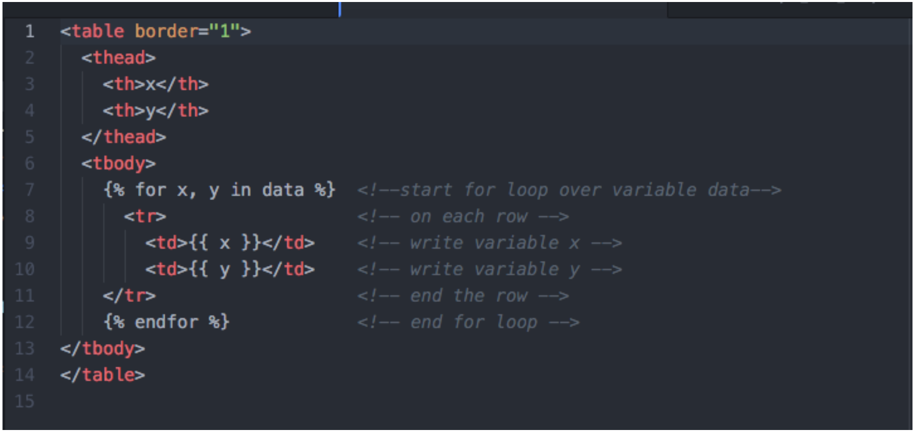


```python
from flask import Flask, render_template
from random import random
app = Flask(__name__)


@app.route('/')
def index():
    n = 100
    x = range(n)
    y = [random() for i in x]
    return render_template('table.html', data=zip(x, y))


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)
```

Templates



```
<table border="1">
  <thead>
    <th>x</th>
    <th>y</th>
  </thead>
  <tbody>
    {% for x, y in data %}  <!--start for loop over variable data-->
      <tr>                  <!-- on each row -->
        <td>{{ x }}</td>    <!-- write variable x -->
        <td>{{ y }}</td>    <!-- write variable y -->
      </tr>                 <!-- end the row -->
    {% endfor %}            <!-- end for loop -->
</tbody>
</table>
```

Figure 11:Flask application with template

Variables

Method *flask.render_template(template_name_or_list, context)* accepts context – the variables that should be available in the template.

- render_template('table.html', data=zip(x, y))
- render_template('hello.html', name=name)

From inside templates you can access *request* and *session* objects

- request.form['username']
- request.args.get('key', '')
- request.cookies.get('username')
- session['username']