Part 3: Python libraries and the Flask web framework

Recap

Python has a rich ecosystem of libraries and frameworks that have been written to solve common requirements.

Packages can be installed via the pip package manager:

```
$ pip install some-package-name
```

And can then be imported as modules in python scripts:

```python
from flask import Flask

app = Flask(__name__)
```

Python also has built in modules that don't need to be installed via pip (e.g. datetime).

Web Applications

Refers to any computer program that contains dynamic logic and deals with user-generated data, which users interact with via the World Wide Web (either through their browser or other client software). This could be

- A website with lots of dynamic content
- A desktop app that interfaces with the web (e.g. most messaging apps)

Web apps typically rely upon a client-server architecture where the logic and content is provided by an application running on a server, but is displayed and runs in a user's web browser (the client).

Flask

Flask is a popular web application framework that provides features such as:

- Running a local server

```python
from flask import Flask

app = Flask(__name__)

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

```
$ flask run
```

Flask (continued)

- HTTP Request routing

```python
@app.route('/books')
def get_books():
    # Code to fetch all book entries from the database and return their details.

@app.route('/books/<id>')
def get_book(id):
    # Code to fetch the book entry with matching id from the database and return its details.

@app.route('/books', methods=['POST'])
def add_book():
    # Code to create a new book entry in the database.
```

Flask (continued)

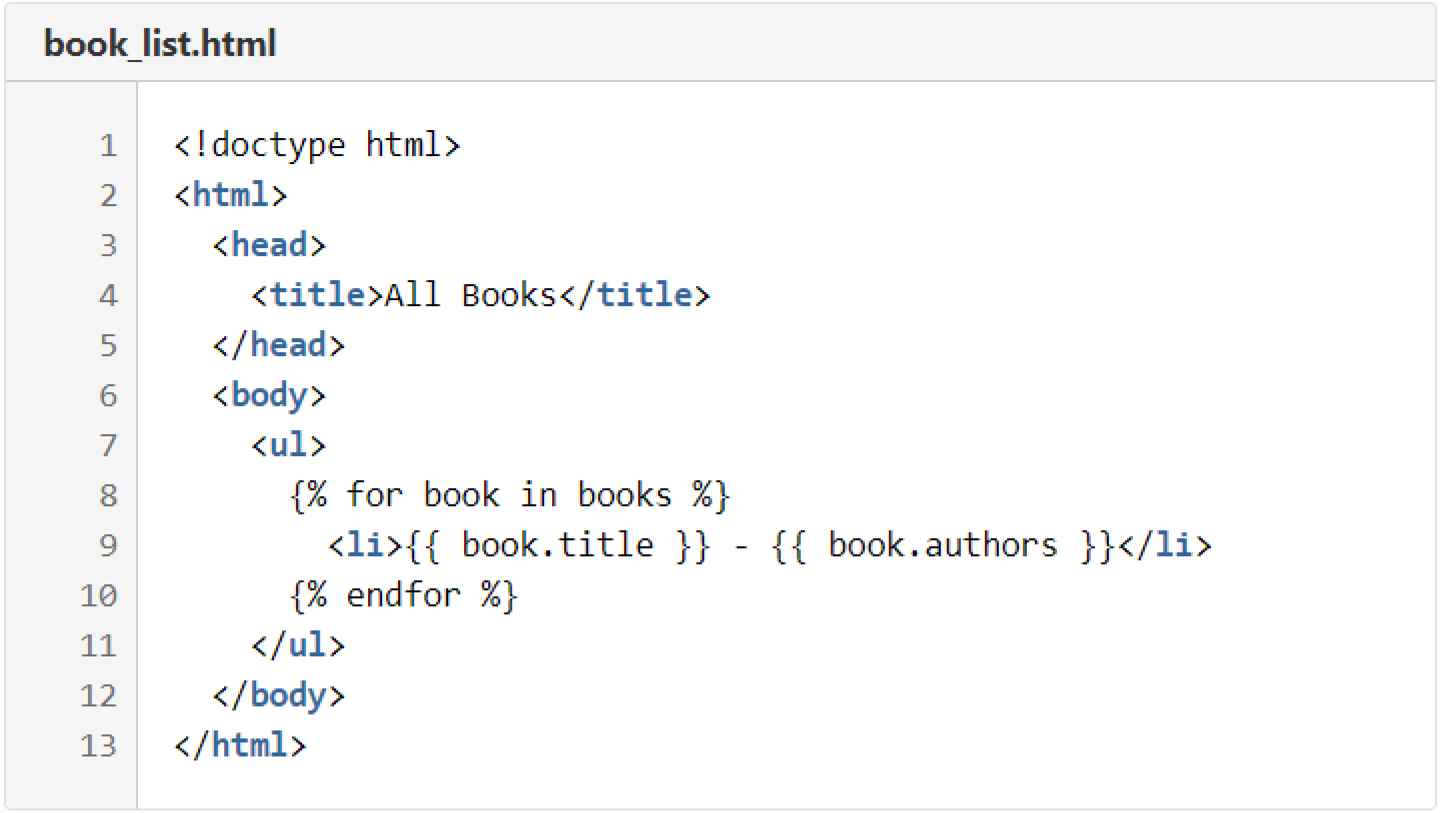
- Templating

```
books = [
  { 'id': 1, 'title': 'Clean Code', 'authors': 'Robert C. Martin' },
  { 'id': 2, 'title': 'The DevOps Handbook', 'authors': 'Gene Kim, Jez Humble, Patrick Debois, John Willis' },
  { 'id': 3, 'title': 'The Phoenix Project', 'authors': 'Gene Kim, Kevin Behr, George Spafford' }
]
```

```
@app.route('/books')
def get_books():
    books = get_all_books_from_db() # Some function that returns the list of book entries from the database.
    return render_template('book_list.html', books=books)
```

Flask (continued)

- Templating



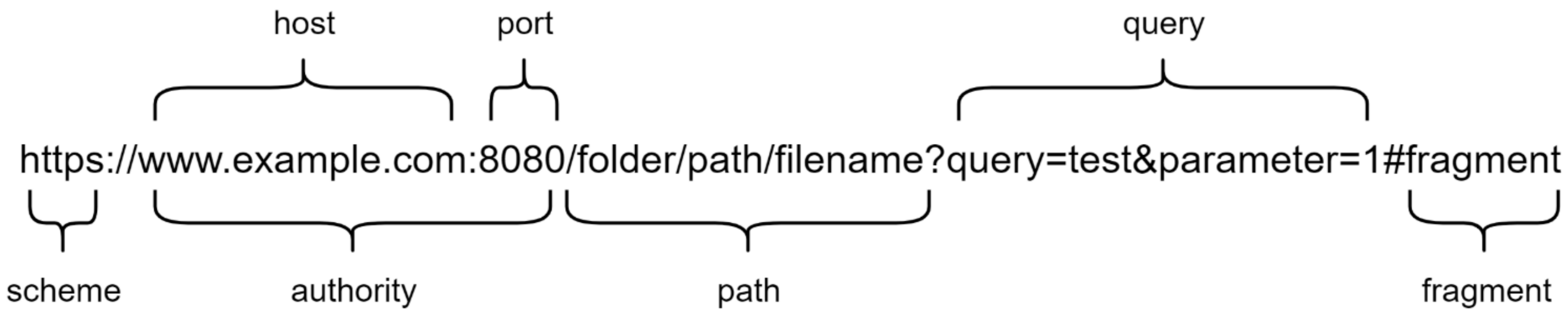
book_list.html

```
<!doctype html>
<html>
  <head>
    <title>All Books</title>
  </head>
  <body>
    <ul>
      {% for book in books %}
        <li>{{ book.title }} - {{ book.authors }}</li>
      {% endfor %}
    </ul>
  </body>
</html>
```

Part 3: New Concepts

Handling Query Parameters

Recall that query parameters are part of the anatomy of a URL:


host
port
query
https://www.example.com:8080/folder/path/filename?query=test&parameter=1#fragment
scheme
authority
path
fragment

They are often used as parameters to HTTP GET requests

Reading Query Parameters In Flask

```python
from flask import Flask, request

app = Flask(__name__)

@app.route("/query")
def query():

    if request.args:
        # We have our query string nicely serialised as a Python dictionary
        args = request.args

        # We'll create a string to display the parameters & values
        serialised = ", ".join(f"{k}: {v}" for k, v in request.args.items())

        # Display the query string to the client in a different format
        return f"(Query) {serialised}"

    else:
        return "No query string received"
```

Template Includes and Extends

The templating library for Flask, Jinja, has various mechanisms for composing HTML templates together.

The simplest is a direct inclusion of one template within another. For example you could have:

```
{% include 'header.html' %}
    Body
{% include 'footer.html' %}
```

Which includes 2 external templates corresponding to the head and footer respectively.

Please note that included templates can also use the same templated parameters as the master template.

Template Includes and Extends (continued)

Corndel Digital.

It is also possible for one template to “inherit” from another template using the `extends` keyword:

Base Template

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
</body>
</html>
```

Child Template

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block content %}
    <h1>Index</h1>
    <p class="important">
      Welcome to my awesome homepage.
    </p>
{% endblock %}
```

This can be useful for sharing a common page structure across a website.

Part 3: Exercise

The Goal

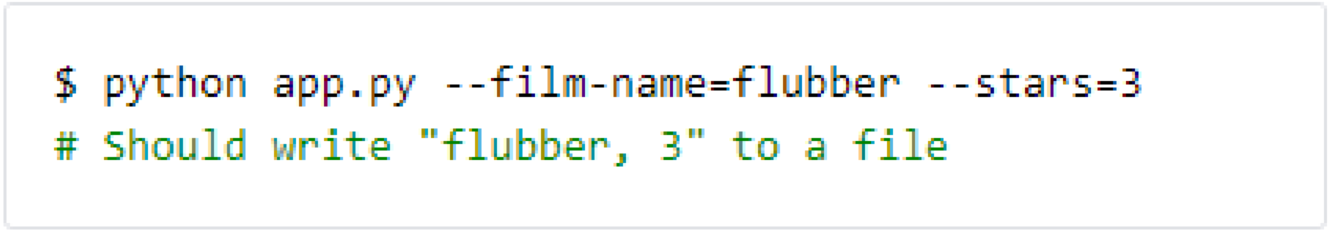
Installing a command line library Click and setting up a command line interface (CLI)

Installing the Flask library and setting up some endpoints covering the use of:

- Static pages
- Templating
- Post form submissions

Step 1:

Install the Click python library and use it generate a command line app for submitting film reviews:



```
$ python app.py --film-name=flubber --stars=3
# Should write "flubber, 3" to a file
```

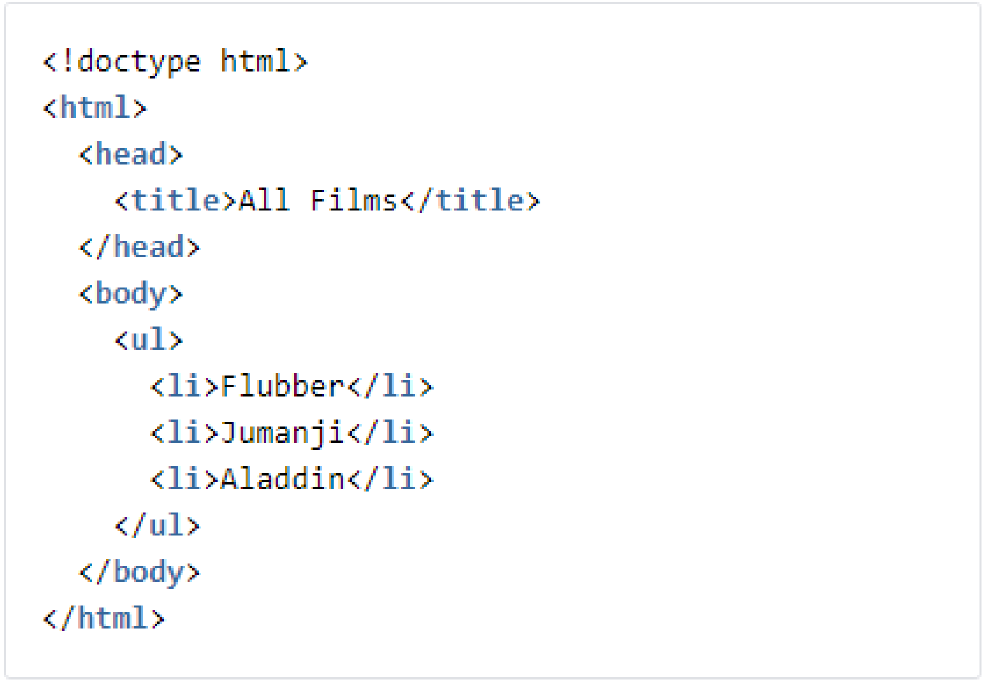
The app should support submitting multiple reviews (by repeatedly running python app.py)

Please show the running app to the tutor to confirm everything is working

Step 2:

Next install the Flask python library and setup a route that serves the following static HTML page:

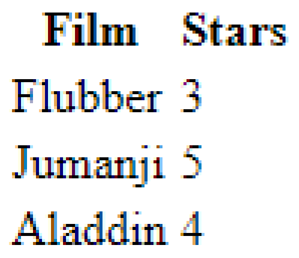
You should serve this under the route "/films/list" on your flask web server.



```html
<!doctype html>
<html>
  <head>
    <title>All Films</title>
  </head>
  <body>
    <ul>
      <li>Flubber</li>
      <li>Jumanji</li>
      <li>Aladdin</li>
    </ul>
  </body>
</html>
```

Step 3:

Next serve a templated table of films along with their star ratings on the route "films/table“:

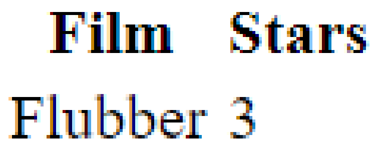


Film	Stars
Flubber	3
Jumanji	5
Aladdin	4

The films should be read from the file generated in Step 1.

Step 4:

Now add support for filtering this list by star rating, e.g. "films/table?stars=3" should return just:



Film	Stars
Flubber	3

At this point you should show the tutor your progress to confirm that everything is working as expected

Step 5:

Finally build a HTTP form served on the route "/films/submit" that fulfils the function of step 1.

If you’ve got this working feel free to experiment with the include/excludes templating structures introduced today or play around with the features listed on the Jinja Template Designer Documentation.