Python Programming

Unit 7 Lab: Flask

Overview

<<<<<<< HEAD:unit-6-flask/instructor-resources/07-flask-unit-lab/README.md

Welcome to the Flask unit lab! You're going to build on to your Rotten Tomatoes application. You'll leave what's there, but you'll also create a web app where a user can enter and search for a movie, then see the results and details.

Welcome to the Flask unit lab! You're going to build on to your Rotten Tomatoes application. You'll leave what's there, but you'll also create a web app where a user can enter a movie to search for, see the results, and see the details.

main:unit-6-flask/instructor-resources/07-flask-unit-lab/unit-lab-6-directions.md

---

Deliverables

You're going to continue building this locally from the last lab. You'll write all of your code in the same `movie_app.py` file.

Run the file from the command line to check your work. You should be able to use the command line argument `flask` to specifically start the Flask app:

```
# This runs the solution from the previous unit lab.
python movie_app.py

# This runs the Flask app!
python movie_app.py flask
```

Requirements

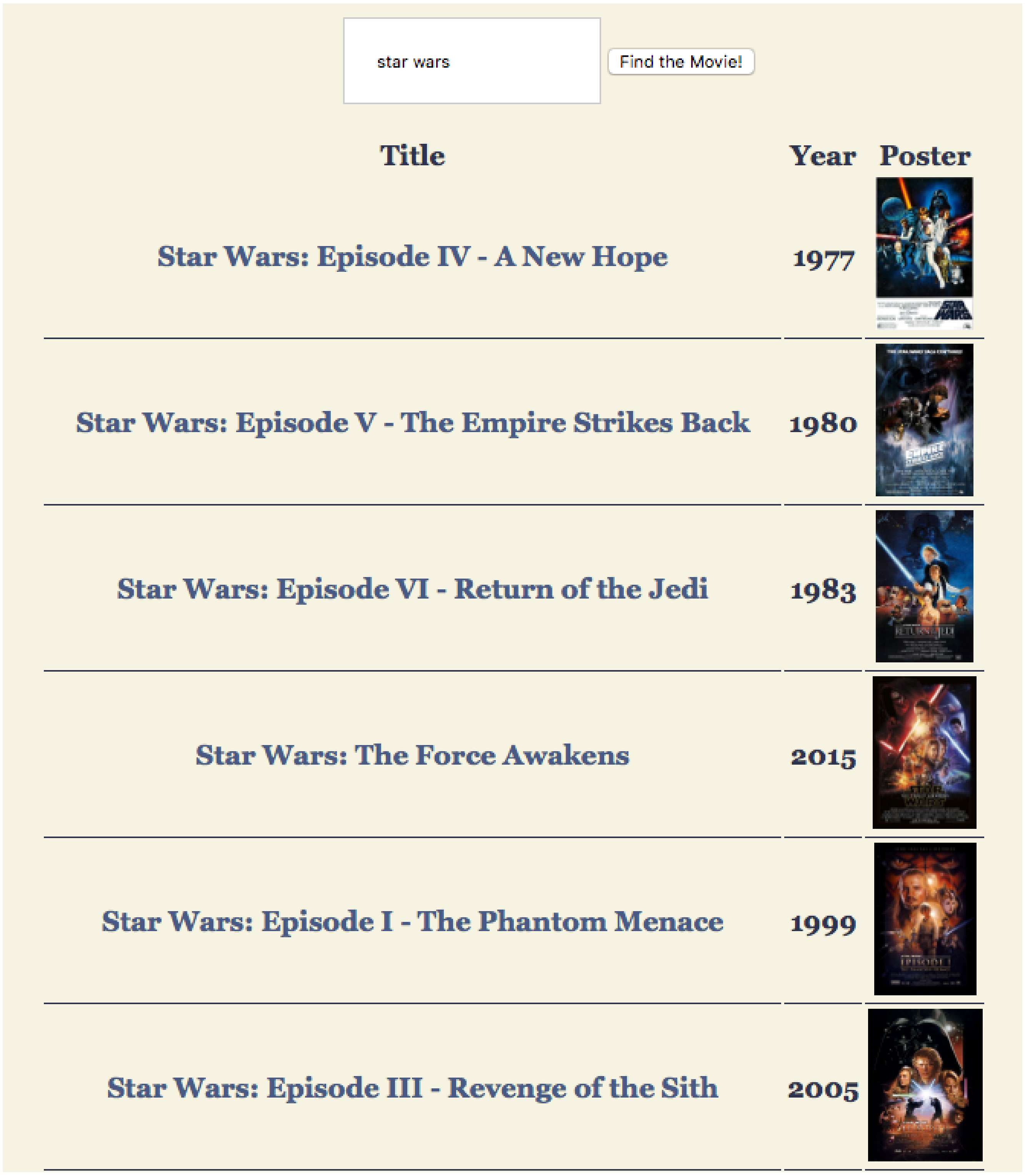
Your program will:

- Be accessible by a local web browser.
- Use the Flask microframework for a user interface.
- Leverage the code already written in previous labs to query the OMDb API.

Your program has three main components: - Landing page. - Search results page. - Movie details page.

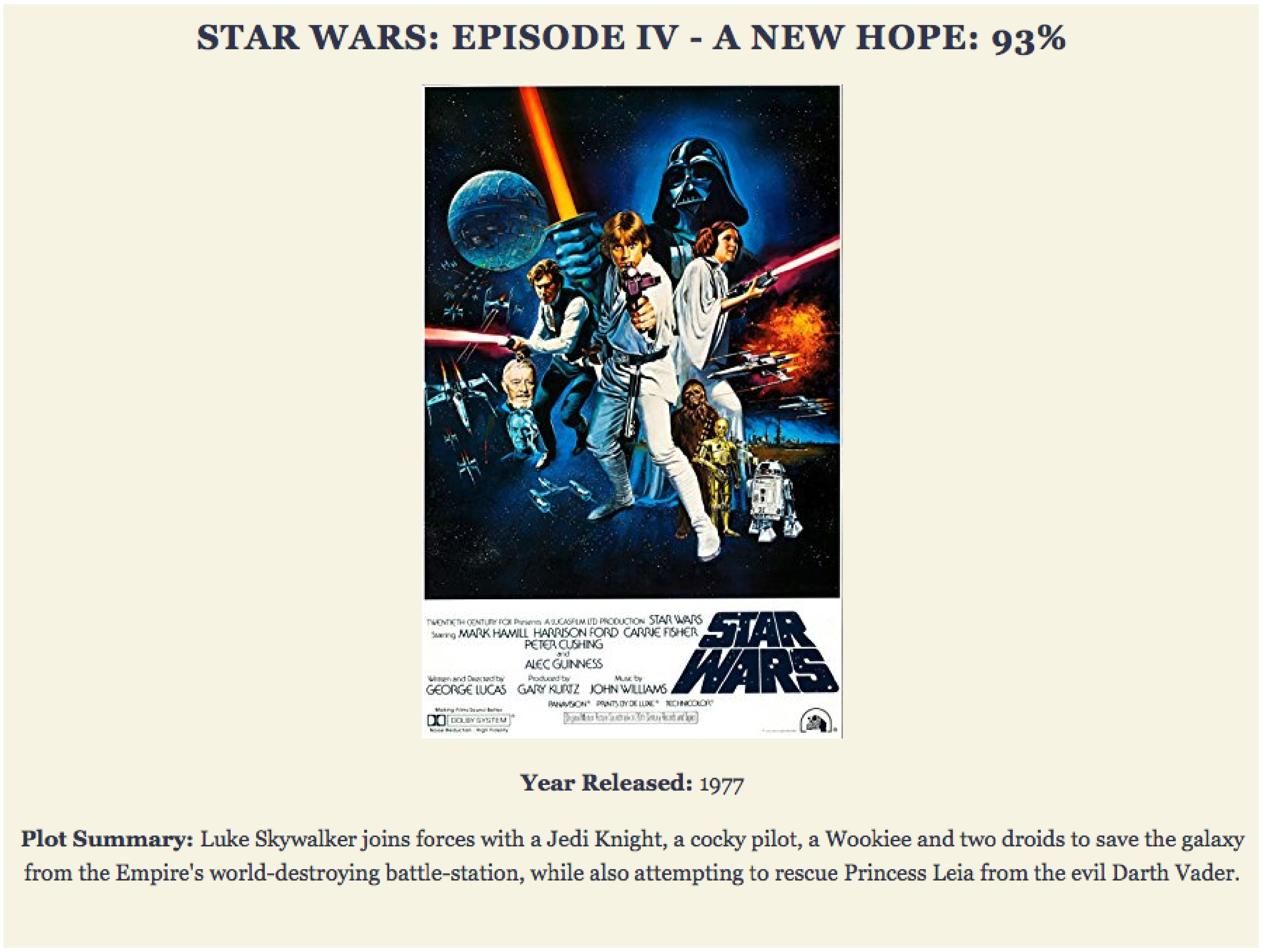
Your program's functionality should look like this (with variation on styling, but it needs some styling):

1. Search Results



Title	Year	Poster
Star Wars: Episode IV - A New Hope	1977	
Star Wars: Episode V - The Empire Strikes Back	1980	
Star Wars: Episode VI - Return of the Jedi	1983	
Star Wars: The Force Awakens	2015	
Star Wars: Episode I - The Phantom Menace	1999	
Star Wars: Episode III - Revenge of the Sith	2005	

2. Movie Details



STAR WARS: EPISODE IV - A NEW HOPE: 93%

Year Released: 1977

Plot Summary: Luke Skywalker joins forces with a Jedi Knight, a cocky pilot, a Wookiee and two droids to save the galaxy from the Empire's world-destroying battle-station, while also attempting to rescue Princess Leia from the evil Darth Vader.

Directions

Augment the `movie_app.py` code you wrote for the Unit 5 lab.

Setting Up a Basic App

1. Click here for the starter CSS file and here for the `home.html` file. Be sure to place them in a `static` and `template` folder appropriately.
2. We're going to have your app be optionally run either as it currently is or as a Flask app. To clarify which a user wants, rename your `main` function to `cli_app()` — "cli" stands for "command line interface" (this is a common programming term — it means your terminal!). Change the function call in the `if __name__` at the bottom of your file, too. Test it out!
3. Now, let's set your program up to run either the Flask web app or the existing command line interface app.
   - Import the `sys` module at the top of your file; this will allow your app to see if the user added any arguments in the command line when they run the app.
   - At the bottom of your lab, it says:

     ```
     # This line tells Python to run main when it first opens.
     if __name__ == "__main__":
       main()
     ```

     Instead, change that to: `# Define the Flask app starting point.`

```
def flask_app():
print("In Flask app")

if __name__ == "__main__":
  # Check command line argument for "flask" and run Flask app if so.
  # Fun fact - these are *args!
  if len(sys.argv) > 1 and sys.argv[1] == "flask":
      flask_app()
  else:
      # If there was no "flask" arg when the program was run,
      # Let the user know.
      print('Run "python movie_app.py flask" for the Flask app.')
      # Run the regular app.
      cli_app()
```

Test it out — run your program with `python movie_app.py`. It should be your normal app.

Then, try running `python movie_app.py flask`. Your `print` statement should appear, instead!

> Note: This is another example of how `print` statements can be helpful — use them as placeholders to make sure a function or conditional block was successfully called.

Let's continue just by making the app appear.

1. Import the correct pieces from `flask` to run Flask, a template, and a `GET` request.
2. Don't forget to initialize your app at the top of the file!
3. Below all the existing code for the `cli` app (but above the `__main__` condition), create an `@app.route` to handle user traffic to the index page, which we'll call `home`.
4. On this page, add a `return` string of your choosing — it's just a test for now.
5. Let's set it up to run. Below, where you have: `# Define the Flask app starting point.`
   ```
   def flask_app():
     print("In Flask app")
   ```
   Change the `print` statement to instead run the app using `app.run(debug=True)`.

Try running the app! Remember, it'll be at `http://127.0.0.1:5000/`.

Making the App More Complex: Searching

This page should allow a user to search and display the results.

1. Using `list_search_results()` as a guide, have your app prompt for user input and display the **first** search result's title.
   - **Hint**: Try something like `matching_movie_list[0]["Title"]`.
2. Be sure to test this functionality!
   - Because we don't have a way to ask the user for input in the app yet, your command line will prompt you for a movie once you've loaded the page.
3. Let's set up the actual webpage. We'll eventually prompt the user for input in the actual webpage, as well as display all the results. This has already been set up for you in the `home` template you downloaded earlier.
   - You don't need to change this template, but you do need to understand it!
   - Start out by testing rendering only the template without the results listed. Compare what you see to the template — is it right? What happens when you click the "Find the Movie!" button?
4. The template's prepared with the search functionality. In your Python app, delete the call for a user's input and instead add a line like this: `# Get the search query from the form. movie_query = request.args.get("query", "")`
   - Don't forget to change the parameters of your `home` app.
   - Have the OMDb search run *only* if `movie_query` exists. Either way, render the template.
   - If you test it out, don't forget to print it!
5. The template is able to print out both the movie title and the search results, if they're passed to it. It takes in:
   - `query`, which is what the user searched for.
   - `results`, which is the matching movie list returned from the search.

Test this out! You should have a form that takes in a movie title and displays the results.

Awesome!

Making the App More Complex: Displaying Movie Details

You can see that each movie title in the search results is hyperlinked, but the link is broken. Let's fix that — we'll add the page and display movie details on it.

1. Create a new template for a movie detail page, `movie.html` (you can use the `home.html` as an outline!). In it:
   - Title the page "Movie Search."
   - Display the title and rating as an `h2`. `title` and `rating` will be values the template receives from

your Python script.
    - Beneath that, put an image link for the poster. The image source here will be `poster`, another value received from your Python script.
    - Beneath that, make a paragraph (`p`) that says "Year Released:" and the `year`, which will be a value received from your Python script.
    - Below that, have a new paragraph with "Plot Summary:", then `plot`, which will be a value received from the Python app. Wrap the "Plot Summary:" and "Year Released" text in a `strong` opening and closing tag.
    - Don't forget to close all tags!
2. Now, we have the template. Back in your Python app, create a route to `/movies/<id>` called `movie`, which takes an `id` value. First, try rendering the template with hard-coded values of your choosing. Test it out!
3. We have our list of search results, each of which has an `id` value identifying it uniquely on Rotten Tomatoes. To get more details about a specific movie, we should search for that specific `id`. Searching for a movie result seems like something that should go in the `OMDb` class!
    - In that class, create a new method, `get_movie_by_id`. You can copy (don't delete) your existing `get_movie()` method, but change the parameter to `id` and argument in the API call to `i=id`.
    - How did we know to do that? It's in the OMDb API docs! Go here and scroll down to "Parameters."
4. Back to your `movies` route, get the API key, create the OMDb object, and then call your new `get_movie_by_id()` method, passing it the `id` parameter.
    - From here, all the values are still hard-coded, but you can test it out to be sure that the API call works!
5. Now, display the actual movie information. Change what's passed into your template to be actual values from the `movie` object.
    - **Hint**: Check your `Movie` class to see how to get information!
    - **Hint**: You'll have to call the `get_movie_rating()` method to specifically get the Rotten Tomatoes rating.

Try it out! Try a bunch of different movies to see your app in action.

Great job!

Making the App a Little More Colorful

Your app is functional, but a little plain. Modify the CSS file your app has to be a little more fun! - **Note**: You probably haven't seen some of the CSS that's currently there! Leave it for now, but if you have time after adding your own styling, play around with it to see what changes.

If you'd like inspiration, here is what we have to make the images at the top of the page. All of it applies to the `body`: - A `font-family` of `"Georgia, sans-serif"`. - (Use the Georgia font if possible. Otherwise, use a default sans-serif font.) - A `color` of `#333a56`. - This changes the text color. - `text-align` to `center` - `font-size` to `16px`. - `background: #f7f5e6`.

We also added styling to the `h2`: - `text-transform` to `uppercase`. - This makes the text all capitals. - `font-weight` to `bold`. - This bolds the text.

Once you're finished styling, you're done! Amazing job.

If you have extra time, play around with the styling or Check out the OMDb docs to see what else you can display. Can you display a message if the Rotten Tomatoes score is above or below a certain score?