CIS 008 Final Project: MovieREC

By: Colin Zejda

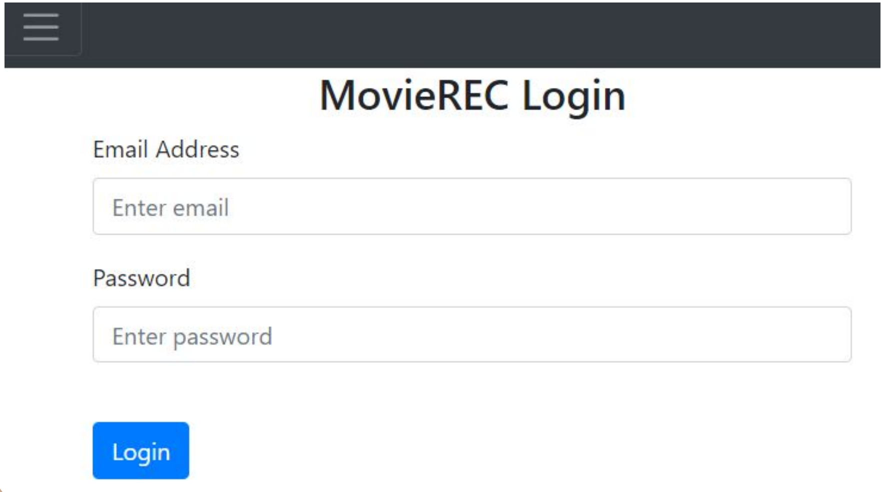
Who loves movies?

- 99% of the population!
- What does MovieREC do?
  - Give movie recommendations!
  - input: movie name or keyword
  - Output: 10 new movie names+links

What else can you find on MovieREC?

- A movie journal
- The current top 10 IMDB movies
- Links and posters of all displayed movies

What does MovieREC look like?



MovieREC Login

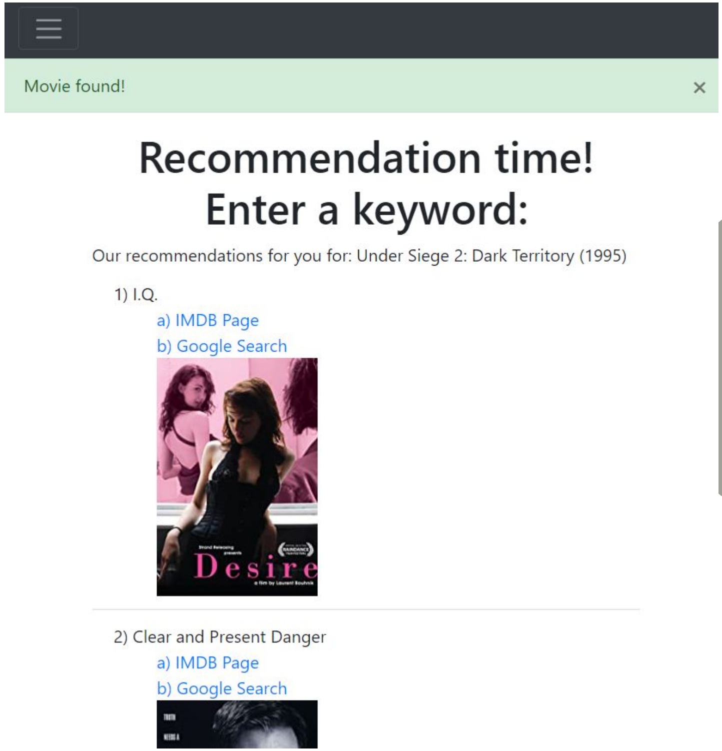
Email Address

Enter email

Password

Enter password

Login



Movie found!

Recommendation time!
Enter a keyword:

Our recommendations for you for: Under Siege 2: Dark Territory (1995)

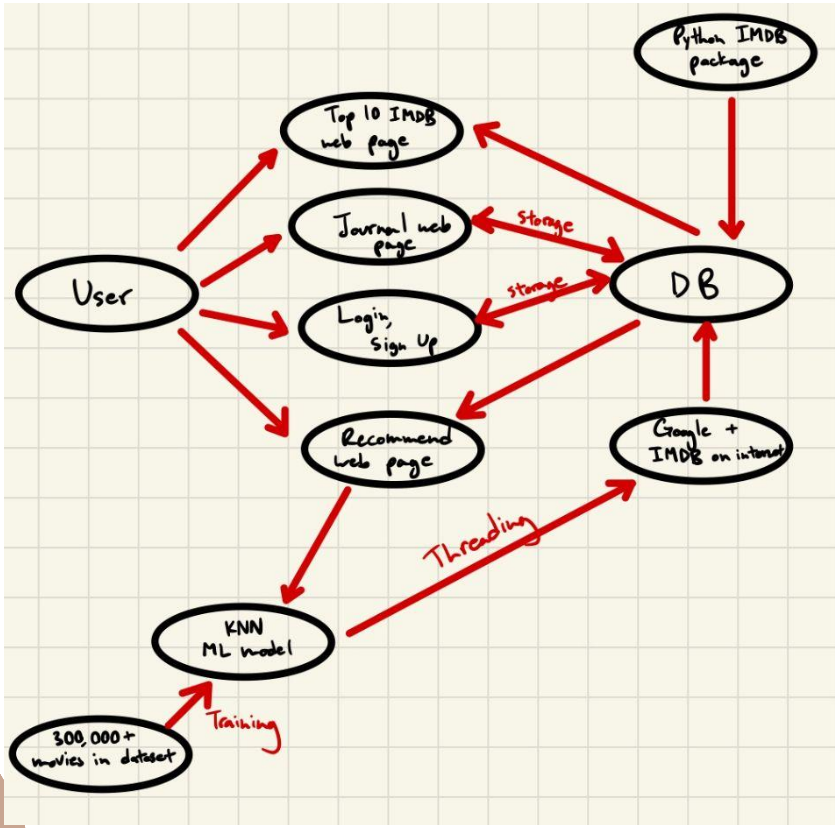
1) I.Q.
 a) IMDB Page
 b) Google Search

2) Clear and Present Danger
 a) IMDB Page
 b) Google Search

Project Architecture

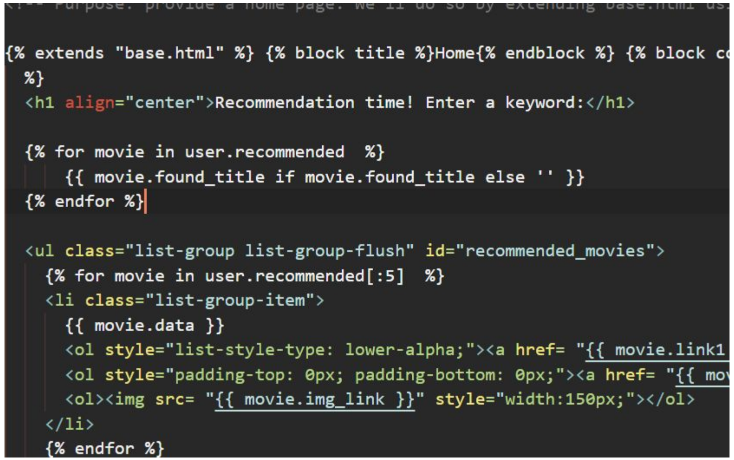
- HTML pages
  - What the user sees, entry of data, login/out
- Database
  - Stores all data
- Python files
  - NN, trained offline
  - Web scraper
  - Blueprints, login/out logic, POST+GET request handling for all HTML files
  - Jinja

Project Arch. Block Diagram



HTML pages

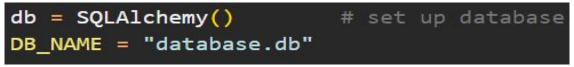
- Login, Sign-up, Recommend, Top-10, Journal
- Jinja
  - Python code in HTML file to access DB
  - Templates rendered by auth.py and views.py
- Bootstrap decoration



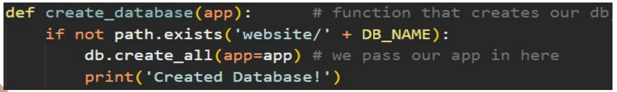
```
<!-- Purpose: provide a home page. We'll do so by extending base.html us

{% extends "base.html" %} {% block title %}Home{% endblock %} {% block co
  %}
  <h1 align="center">Recommendation time! Enter a keyword:</h1>

  {% for movie in user.recommended  %}
      {{ movie.found_title if movie.found_title else '' }}
  {% endfor %}

  <ul class="list-group list-group-flush" id="recommended_movies">
    {% for movie in user.recommended[:5]  %}
    <li class="list-group-item">
      {{ movie.data }}
      <ol style="list-style-type: lower-alpha;"><a href= "{{ movie.link1
      <ol style="padding-top: 0px; padding-bottom: 0px;"><a href= "{{ mov
      <ol><img src= "{{ movie.img_link }}" style="width:150px;"></ol>
    </li>
    {% endfor %}
```

The DB

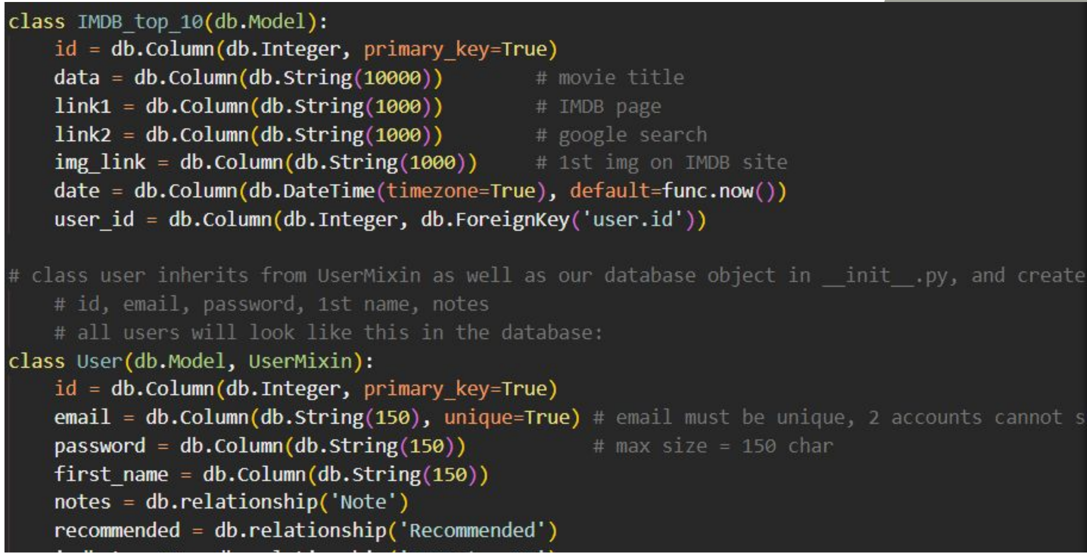
- Made with SQL-alchemy
  - Works well with Flask
- DB created in __init__.py, tables and their relationships created in models.py



```
db = SQLAlchemy()          # set up database
DB_NAME = "database.db"
```

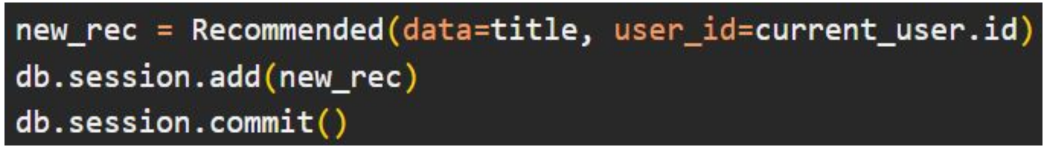


```
def create_database(app):       # function that creates our db
    if not path.exists('website/' + DB_NAME):
        db.create_all(app=app) # we pass our app in here
        print('Created Database!')
```

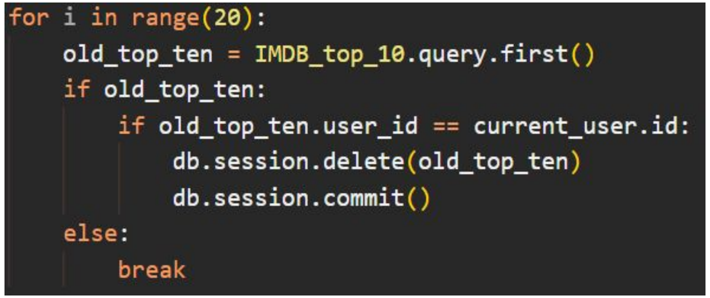


```
class IMDB_top_10(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    data = db.Column(db.String(10000))        # movie title
    link1 = db.Column(db.String(1000))        # IMDB page
    link2 = db.Column(db.String(1000))        # google search
    img_link = db.Column(db.String(1000))     # 1st img on IMDB site
    date = db.Column(db.DateTime(timezone=True), default=func.now())
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))

# class user inherits from UserMixin as well as our database object in __init__.py, and create
    # id, email, password, 1st name, notes
    # all users will look like this in the database:
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True) # email must be unique, 2 accounts cannot s
    password = db.Column(db.String(150))          # max size = 150 char
    first_name = db.Column(db.String(150))
    notes = db.relationship('Note')
    recommended = db.relationship('Recommended')
```

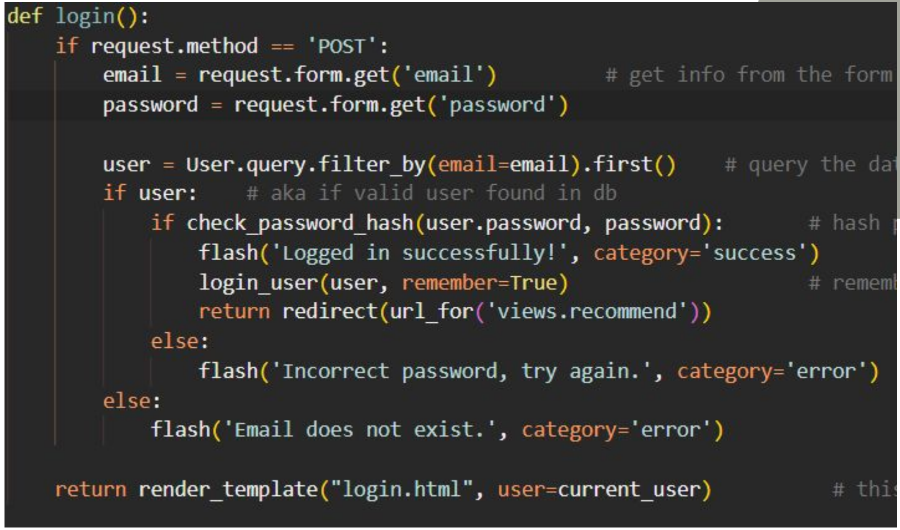
Interacting with the DB



```python
new_rec = Recommended(data=title, user_id=current_user.id)
db.session.add(new_rec)
db.session.commit()
```



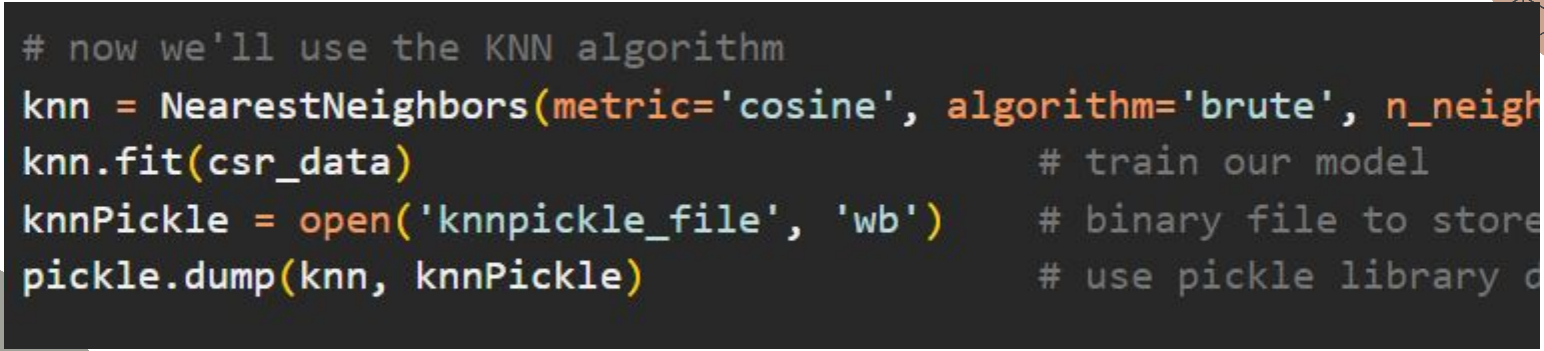
```python
for i in range(20):
    old_top_ten = IMDB_top_10.query.first()
    if old_top_ten:
        if old_top_ten.user_id == current_user.id:
            db.session.delete(old_top_ten)
            db.session.commit()
    else:
        break
```



```python
def login():
    if request.method == 'POST':
        email = request.form.get('email')          # get info from the form
        password = request.form.get('password')

        user = User.query.filter_by(email=email).first()    # query the dat
        if user:    # aka if valid user found in db
            if check_password_hash(user.password, password):       # hash p
                flash('Logged in successfully!', category='success')
                login_user(user, remember=True)                      # rememb
                return redirect(url_for('views.recommend'))
            else:
                flash('Incorrect password, try again.', category='error')
        else:
            flash('Email does not exist.', category='error')

    return render_template("login.html", user=current_user)          # thi
```

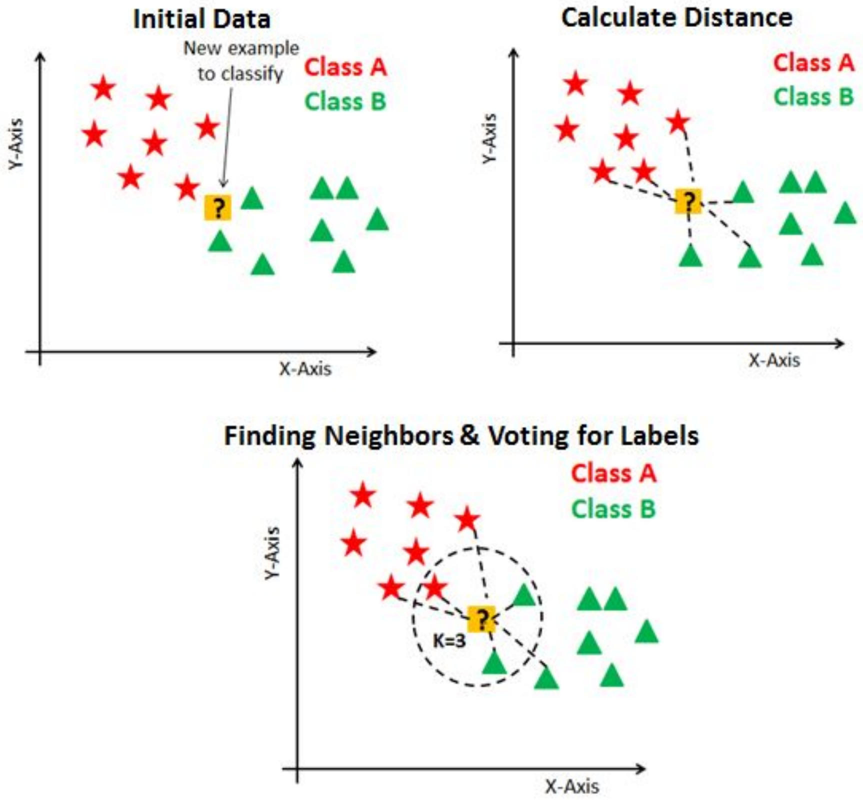
How do we recommend movies?

- DB with 300,000+ movies with user ratings
- Trained KNN machine learning model on this dataset
  - Separate training and usage of model (pickle to new file)

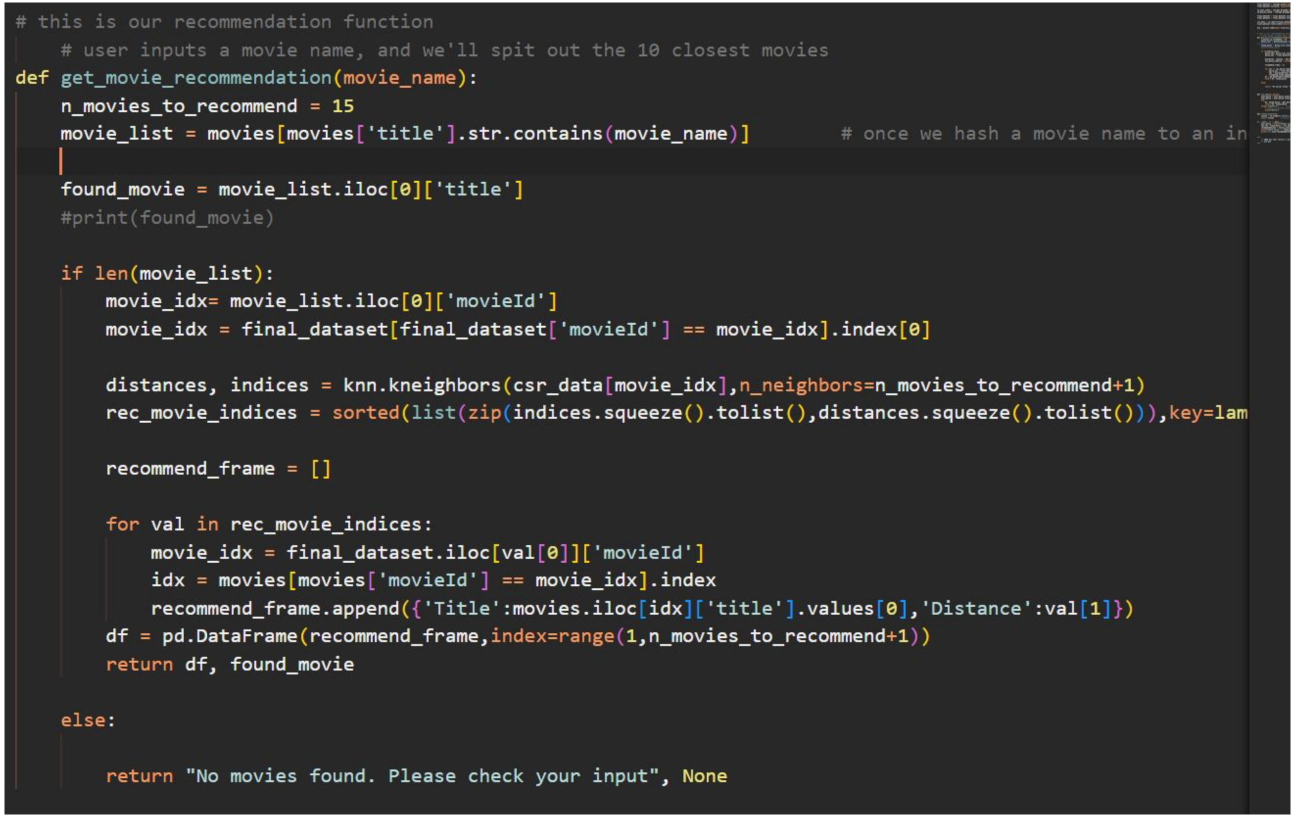


```
# now we'll use the KNN algorithm
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neigh
knn.fit(csr_data)                              # train our model
knnPickle = open('knnpickle_file', 'wb')       # binary file to store
pickle.dump(knn, knnPickle)                    # use pickle library d
```

KNN recommendations



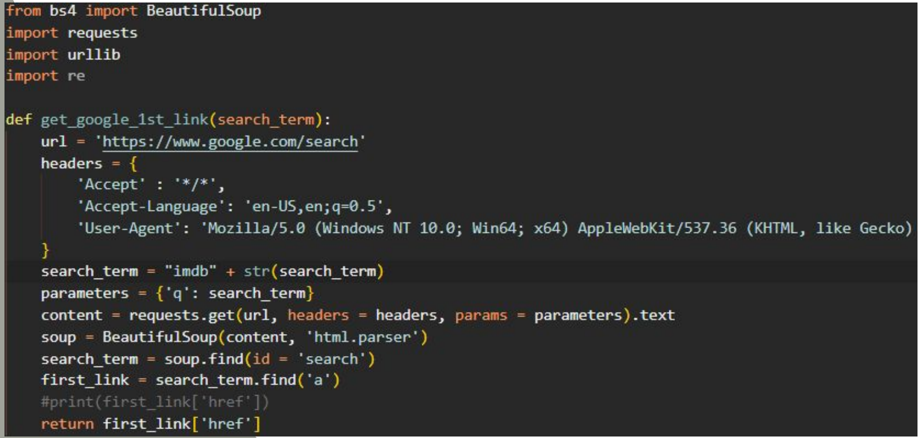
How do we recommend movies?



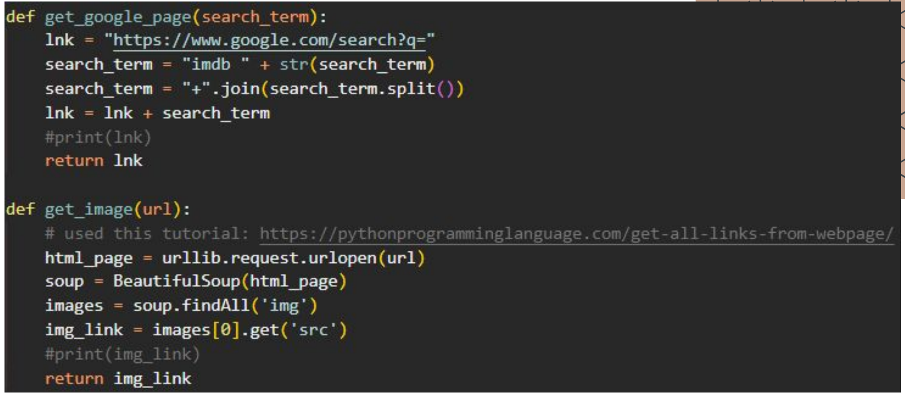
```python
# this is our recommendation function
    # user inputs a movie name, and we'll spit out the 10 closest movies
def get_movie_recommendation(movie_name):
    n_movies_to_recommend = 15
    movie_list = movies[movies['title'].str.contains(movie_name)]        # once we hash a movie name to an in

    found_movie = movie_list.iloc[0]['title']
    #print(found_movie)

    if len(movie_list):
        movie_idx= movie_list.iloc[0]['movieId']
        movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]

        distances, indices = knn.kneighbors(csr_data[movie_idx],n_neighbors=n_movies_to_recommend+1)
        rec_movie_indices = sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),key=lam

        recommend_frame = []

        for val in rec_movie_indices:
            movie_idx = final_dataset.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index
            recommend_frame.append({'Title':movies.iloc[idx]['title'].values[0],'Distance':val[1]})
        df = pd.DataFrame(recommend_frame,index=range(1,n_movies_to_recommend+1))
        return df, found_movie

    else:

        return "No movies found. Please check your input", None
```

Beautiful Soup

- Web scraping → get the top result off of Google
  - Then again to get the IMDB movie poster
  - Href for <a>, src for <img>



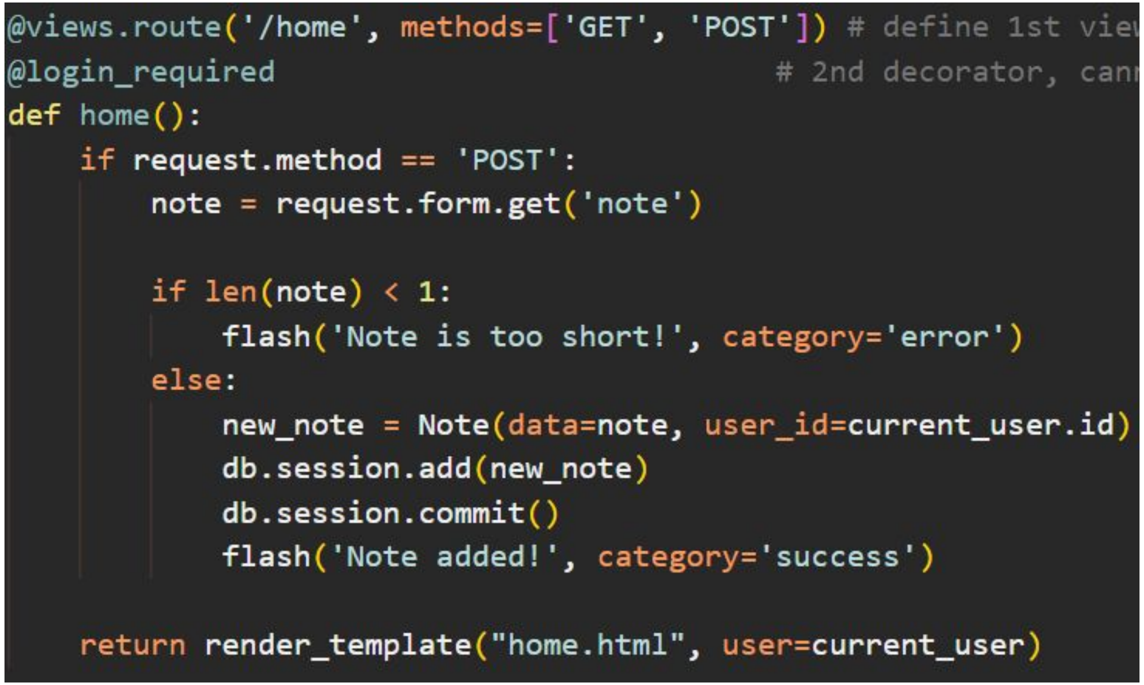
```python
from bs4 import BeautifulSoup
import requests
import urllib
import re

def get_google_1st_link(search_term):
    url = 'https://www.google.com/search'
    headers = {
        'Accept' : '*/*',
        'Accept-Language': 'en-US,en;q=0.5',
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    }
    search_term = "imdb" + str(search_term)
    parameters = {'q': search_term}
    content = requests.get(url, headers = headers, params = parameters).text
    soup = BeautifulSoup(content, 'html.parser')
    search_term = soup.find(id = 'search')
    first_link = search_term.find('a')
    #print(first_link['href'])
    return first_link['href']
```



```python
def get_google_page(search_term):
    lnk = "https://www.google.com/search?q="
    search_term = "imdb " + str(search_term)
    search_term = "+".join(search_term.split())
    lnk = lnk + search_term
    #print(lnk)
    return lnk

def get_image(url):
    # used this tutorial: https://pythonprogramminglanguage.com/get-all-links-from-webpage/
    html_page = urllib.request.urlopen(url)
    soup = BeautifulSoup(html_page)
    images = soup.findAll('img')
    img_link = images[0].get('src')
    #print(img_link)
    return img_link
```

Some interesting code bits

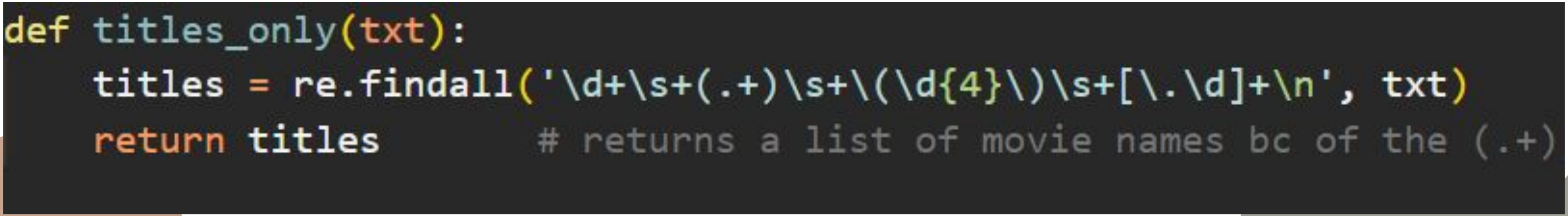
- Decorator usage



```
@views.route('/home', methods=['GET', 'POST']) # define 1st vie
@login_required                                # 2nd decorator, can
def home():
    if request.method == 'POST':
        note = request.form.get('note')

        if len(note) < 1:
            flash('Note is too short!', category='error')
        else:
            new_note = Note(data=note, user_id=current_user.id)
            db.session.add(new_note)
            db.session.commit()
            flash('Note added!', category='success')

    return render_template("home.html", user=current_user)
```

Some interesting code bits

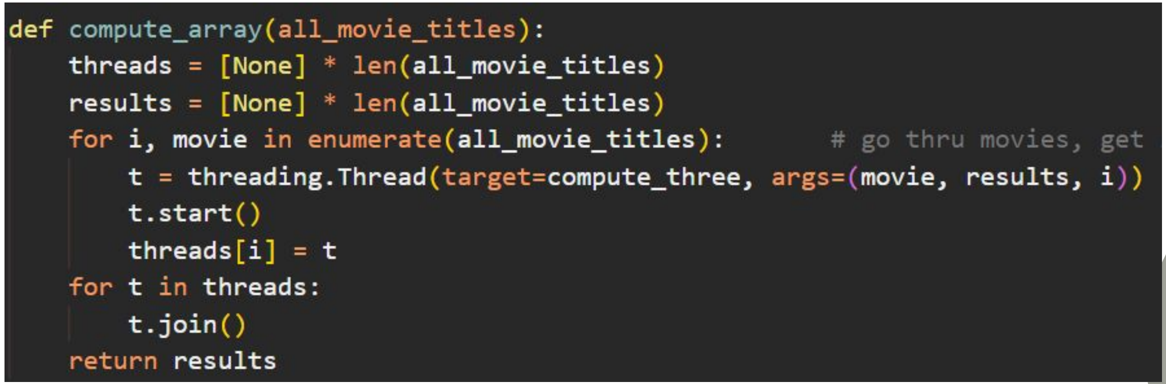
- REGEX usage
    - \d for #'s
    - + means multiple times
    - \s means whitespace
    - (+.) means give back whatever's in the parentheses, here it was the move title
    - \(means find a parenthesis in the expression
    - [] matches any characters inside the brackets



```python
def titles_only(txt):
    titles = re.findall('\d+\s+(.+)\s+\(\d{4}\)\s+[\.\d]+\n', txt)
    return titles        # returns a list of movie names bc of the (.+)
```

Threading

- Web scraping took 5 sec per request for 10 requests
  - Wait took forever
  - Soln: parallelize some of the work using threads



```python
def compute_array(all_movie_titles):
    threads = [None] * len(all_movie_titles)
    results = [None] * len(all_movie_titles)
    for i, movie in enumerate(all_movie_titles):       # go thru movies, get
        t = threading.Thread(target=compute_three, args=(movie, results, i))
        t.start()
        threads[i] = t
    for t in threads:
        t.join()
    return results
```

Now Let's see the site!