```
'''The code is  written in Python using the pandas and numpy libraries.
The Comments in the code are written in blue color'''


import pandas as pd
import numpy as np
from pandas import DataFrame, Series

neg_inf = -100000 #Parameter used in sorting (to set the self similarity of a
                  #       user as well as to set the similarities for users
                  #       who have not rated the movie under consideration,
                  #       so that they are not considered in top-n sorting)
num_neighbours = 30

#Input Users and Movies - List of (User, Movie) tuples
list_inputs = [(1779,557),(1779,13),(1779,862),(1779,857),
        (1779,98),(891,8467),(891,393),(891,745),(891,36658),
        (891,424),(2993,786),(2993,745),(2993,2164),
        (2993,194),(2993,278),(4558,2501),(4558,1572),
        (4558,280),(4558,393),(4558,141),(633,808),(633,809),
        (633,9806),(633,278),(633,114)]
list_inputs.reverse()

#Initializing basic variables
mnames = ['movie_id', 'movie_title']
#used as column headers for loading movies data in next step

#using the csv data loading function from pandas
movies = pd.read_csv('movie-titles.csv',header=None,
                        names=mnames, index_col=0)

rnames = ['user_id', 'movie_id', 'rating']
#used as column headers for loading ratings data in next step

#using the data loading function from pandas
ratings_raw = pd.read_table('ratings.csv', sep=',',header=None,names=rnames)

#Building User Movie Ratings Matrix
#Using the pandas pivot table function to pivot the data
#with rows as users and columns as movies and ratings as values
ratings_mat = ratings_raw.pivot_table('rating',rows='user_id',
                                cols='movie_id',aggfunc='mean')

ratings_mat_ori = ratings_mat.copy()
#Retaining a copy of the original ratings matrix before mean normalizing

#Mean Normalizing the Ratings Matrix
ratings_mat = ratings_mat.sub(ratings_mat.mean(axis=1), axis=0)

#Retaining a mean normalized version,
#without normalizing row magnitudes to 1.0
ratings_unnorm = ratings_mat.copy()

#Normalizing the magnitude of each row of the ratings matrix
#so that cosine similarity can then be computed using a dot product
ratings_norm = ratings_mat.copy().fillna(0)
#Squaring each element and summing elements in each row
ratings_norm = np.square(ratings_norm).apply(np.sum, axis=1)
#Taking the square root to get the magnitude of each row of the ratings matrix
ratings_norm = np.sqrt(ratings_norm)
#Making all row vectors have unit magnitude by dividing by magnitude
ratings_mat = ratings_mat.div(ratings_norm, axis=0)
#Filling all NA values with zeros to facilitate dot product computations
ratings_mat = ratings_mat.fillna(0)

g = open('uucfOutput.txt', 'w') #Opening Output file

#Looping through inputs
for user_id, movie_id in list_inputs:

        #Raw similarity scores computed using a dot product
        #of the ratings matrix and user vector
        suv_raw = ratings_mat.dot(ratings_mat.ix[user_id].transpose())
```

```
        #Setting user's self similarity to negative infinity
        suv_raw.ix[user_id] = neg_inf

        #Setting the similarity scores for users who have
        #not seen movie with id movie_id to negative infinity.
        # np.isnan() function is used to identify the indices
        #of the users who have not seen movie_id
        suv_raw.ix[np.isnan(ratings_mat_ori[movie_id])] = neg_inf

        #The similarity scores vector is sorted and the
        #top 30 (num_neighbors) scores are collected as suv
        suv_raw.sort(ascending=False)
        suv = suv_raw[:num_neighbours]
        #Obtaining the denominator for the weighted average rating computation
        suv_sum = np.sum(suv.copy().apply(np.abs))

        #Computing mean score for user with id user_id
        mean_user_id = ratings_mat_ori.ix[user_id].mean()

        #Obtaining the ratings of only the selected neighbors
        #from the mean normalized ratings matrix
        ratings_suv = DataFrame(ratings_unnorm[movie_id],
                                        index=suv.index).fillna(0)

        #Computing final score for user_id for movie_id
        score = mean_user_id + ratings_suv.transpose().dot(suv)/suv_sum

        #Writing output to file
        g.write('%d,%d,%.4f,%s\n' %(user_id, movie_id, score,
                                movies.ix[movie_id].values[0]))

#Closing Output file
g.close()
```