CISC 121 - Assignment 3A

Written by: Jordan Belinsky

Module 1: Users

def users():

This function represents a database of standard users.

Parameters: none

Returns: a list of dictionaries holding user data

def register_user(name, username, address, email, credit, expiration, password, users, watchlist):

This function appends a user to the users database, checking to verify that the inputted username is unique.

Parameters: name: the user's full name (string)
username: a unique user-made username (string)
address: the user's living address (string)
email: the user's email address (string)
credit: the user's credit/debit card number (integer)
expiration: the user's credit/debit card expiration date (integer)
password: a user-made password (string)
users: a database of all user dictionaries (list)
watchlist: a database of all watched movies in this user (list)

Returns: a dictionary of the user's data, appended into users database

def view_account(username, users):

This function pulls all the user information for their account.

Parameters: username: a unique user-made username (string)
users: a database of all users (list)

Returns: a dictionary of the user's data

def login_user(username, password, users):

This function verifies username and password input to allow a user to log in.

Parameters: username: a unique user-made username (string)
password: a user-made password (string)
users: a database of all user dictionaries (list)

Returns: none

Module 2: Administrators

def administrators():

This function represents a database of users with administrative privileges.

Parameters: none

Returns: a list of dictionaries holding administrator data

def add_administrator(name, username, password):

This function appends a user to the administrators database.

Parameters: name: the administrator's full name (string)
username: a unique username (string)
password: a administrator-made password (string)

Returns: a dictionary of the administrator's data, appended to the administrators database

def add_user(name, username, address, email, credit, expiration, password, users, watchlist):

This function appends a user to the users database, checking to verify that the inputted username is unique.

Parameters: name: the user's full name (string)
username: a unique user-made username (string)
address: the user's living address (string)
email: the user's email address (string)
credit: the user's credit/debit card number (integer)
expiration: the user's credit/debit card expiration date (integer)
password: a user-made password (string)
users: a database of all user dictionaries (list)
watchlist: a database of all watched movies in this user (list)

Returns: a dictionary of the user's data, appended into users database

def remove_user(username, users):

This function removes a user from the users database.

Parameters: username: a unique user-made username (string)
users: a database of all users (list)

Returns: print("The user has been successfully removed.")

def view_user(username, users):

This function pulls all the user information for their account.

Parameters: username: a unique user-made username (string)
users: a database of all users (list)

Returns: a dictionary of the user's data

def update_user(username, key, id, users):

This function allows updating of any of a user's data.

Parameters: username: the user's username (string)
key: the attribute which you wish to update (string)
id: the value which you wish to be changed to
users: a database of all users (list)

Returns: a dictionary of the user's data, appended to the users database

def login_administrator(username, password, administrators):

This function verifies username and password input to allow a administrator to log in.

Parameters: username: a unique username (string)
password: a user-made password (string)
administrators: a database of all administrator dictionaries (list)

Returns: none

Module 3: Movies

def movies():

This function represents a database of movies.

Parameters: none

Returns: a list of dictionaries holding movie data

def add_movie(name, genre, length, poster, movies):

This function appends a movie to the movies database.

Parameters: name: the title of the movie (string)
genre: the genre/category of the movie (string)
length: the runtime of the movie (integer)
poster: an image file of the movie poster (string)
movies: the database of all movies (list)

Returns: a dictionary of the movie's data, appended to the movies database

def movie_search(query, movies):

This function searches through the movie list based on a search input from the user.

Parameters: query: the user-inputted search (string)
movies: the database of all movies (list)

Returns: a list of dictionaries of relevant movies

def genre_search(genre, movies, watchlist):

This function searches through the movie list based on a genre input from the user, and remove watched movies from the listings.

Parameters: genre: the genre/category of the movie (string)
movies: the database of all movies (list)
watchlist: a database of all watched movies in this user (list)

Returns: a list of dictionaries of relevant movies

def display_movie(movies, title, poster):

This function displays a movie as a poster as the user views it.

Parameters: movies: the database of all movies (list)
title: the movie which is being displayed
poster: an image file of the poster (string)

Returns: an image file string, to be displayed

def play_movie(movies, title):

This function plays a user-selected movie, and.

Parameters: movies: the database of all movies (list)
title: the movie which the user selected (string)
Returns: none

def mark_watched(movies, title, watchlist):

This function adds it to the user's watched movies list once it is completed.
Parameters: movies: the database of all movies (list)
title: the movie which the user watched (string)
watchlist: a database of all watched movies in this user (list)
Returns: a dictionary of the movie's data, appended to the watchlist database

def recommend_based_genre(username, users, genre, movies):

This function recommends movies to a user based on their favorite genre in their watchlist.
Parameters: username: the user's username (string)
users: a database of all users (list)
genre: the user's favorite genre (string)
movies: the database of all movies (list)
Returns: a list of dictionaries of relevant movies

def recommend_based_rating(username, users, movies):

This function recommends movies to a user based on ratings similar to movies in their watchlist.
Parameters: username: the user's username (string)
users: a database of all users (list)
movies: the database of all movies (list)
Returns: a list of dictionaries of relevant movies

def recommend_based_length(username, users, movies):

This function recommends movies to a user based on lengths similar to movies in their watchlist.
Parameters: username: the user's username (string)
users: a database of all users (list)
movies: the database of all movies (list)
Returns: a list of dictionaries of relevant movies

Module 4: Reporting

def genre_count(movies, genre):

This function generates a report of the number of movies within a genre.
Parameters: movies: the database of all movies (list)
genre: the genre to check for movies (string)
Returns: an integer value of the number of movies within the given genre

def top10_rating_overall(movies):

This function generates a report of the top 10 highest rated movies overall.
Parameters: movies: the database of all movies (list)
Returns: a list of dictionaries of the top 10 movies overall

def top10_rating_genre(movies, genre):

This function generates a report of the top 10 highest rated movies within a genre.
Parameters: movies: the database of all movies (list)
genre: the genre to check for movies (string)
Returns: a list of dictionaries of the top 10 movies in the given genre

def top5_most_watched(movies, users):

This function generates a report of the top 5 most watched movies overall.
Parameters: movies: the database of all movies (list)
users: a database of all users (list)
Returns: a list of dictionaries of the top 5 most watched movies

def average_rating_genre(movies, genre):

This function generates a report of the average rating within a genre.
Parameters: movies: the database of all movies (list)
genre: the genre to check for ratings (string)
Returns: an integer value of the average rating within the given genre

def length_report(movies):

This function generates a report of the longest, shortest, and average movie length overall.
Parameters: movies: the database of all movies (list)
Returns: a string containing the movie titles and runtimes of the longest, shortest, and average length respectively.