2023

CYSE 250 Python Project

FAMOUS ACTORS AND ACTRESSESDATABASE

KEVIN MAXEY

Personal Computer Specifications:

Computer:

- Brand/Model: [Insert Personal Computer Brand/Model]

- Processor: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz, 3601 Mhz, 8 Core(s), 8 Logical Processor(s)

- RAM: 64 GB

- Storage 1: Samsung SSD 860 EVO 500GB

- Operating System: Windows 10 Home

Network:

- Connection Type: Wireless

Additional Information:

- Programming Language: Python ver. 3.12.0

Problem Statement:
The need for efficient and user-friendly systems to access information about actors and actresses is ever-growing in entertainment databases. Traditional databases often need more interactivity and engagement, which users desire. The challenge lies in developing a system that provides comprehensive details about actors and actresses, enhancing user experience and knowledge retention.

Introduction:
One of the intriguing applications in software systems is the Actor Database System, designed to provide information about actors and actresses, complete with search functionalities and a coming soon quiz feature. This two-page report aims to dissect the components of this system, highlighting the structure and functionality of the database code, server code, and client code.

Database Code:
The backbone of the Actor Database System lies in the actors_database.py file. This Python script encapsulates an extensive collection of information about actors and actresses. The database is organized as a Python dictionary, with actor and actress names as keys, and corresponding details such as age, movies, and country of origin as associated values. This database design allows for efficient retrieval of information given the name of an actor or actress. Dictionaries provide a convenient structure for quick and straightforward access to individual profiles.
Moreover, the database script exhibits an elegant and scalable approach, accommodating the addition of new actors and actresses seamlessly. Each entry is well-structured, comprising essential attributes that contribute to a comprehensive understanding of the artist's career and background. The database script showcases good programming practices, utilizing data structures effectively to represent and manage information systematically.

Server Code:
The server-side implementation, depicted in the server.py script, establishes the communication bridge between the database and clients. The server operates on a simple yet robust socket-based architecture, listening for incoming connections and handling client requests. The server script incorporates error-handling mechanisms to manage unexpected situations, ensuring graceful stability during client interactions.
One key feature of the server code is its responsiveness to various client commands. The handle_client function interprets incoming commands, distinguishing between search queries. In the case of a search query, the server utilizes the search_info function from the database code to retrieve and relay actor or actress information back to the client. The modular structure allows for future expansion, adding new functionalities without compromising existing features.

Client Code:
The client.py script represents the user interface of the Actor Database System. This component engages users through a straightforward menu-based system, offering the option to search for actor or actress information or close the connection. The client code adheres to user-friendly design principles, guiding users through the available functionalities.

The client script interacts with the server by sending commands and receiving corresponding data. Implementing the client code showcases the effective use of the pickle module for serializing and deserializing Python objects, enabling the transmission of complex data structures between the client and server.

Import Modules:

Several essential Python modules facilitate various functionalities in implementing the Actors and Actresses Database System. The json module plays a crucial role in saving the database information to a file in JSON format, allowing for persistent storage and easy retrieval of actor and actress details. Furthermore, the socket and pickle modules are employed in the client-server communication setup. The socket module enables the establishment of network connections. In contrast, the pickle module facilitates the serialization and deserialization of Python objects, allowing seamless data interchange between the client and server components. Collectively, these modules contribute to the functionality, efficiency, and reliability of the Actors and Actresses Database System.

In conclusion, with its meticulously crafted database, server, and client components, the Actor Database System stands as a testament to the fusion of user-friendliness and functionality. The well-designed architecture of each component ensures smooth interactions and lays the foundation for future enhancements.

Appendix A: actors_database.py

actors_database.py - C:\Users\kmaxe\AppData\Local\Programs\Python\Python312\Scripts\actors_database.py (3.12.0)

File Edit Format Run Options Window Help

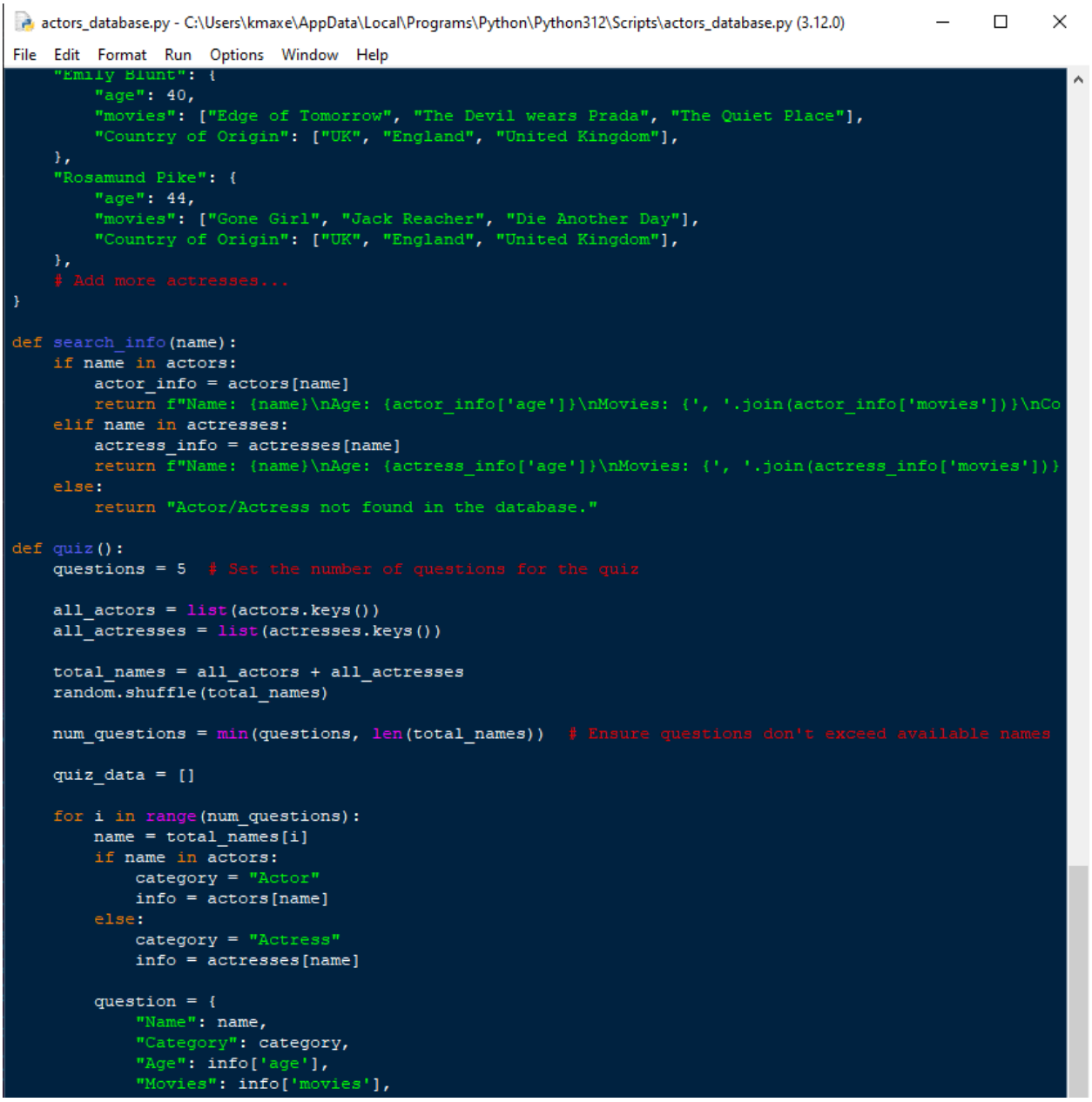
```
# actors_database.py
import random
import json

actors = {
    "Leonardo DiCaprio": {
        "age": 48,
        "movies": ["Inception", "The Revenant", "Titanic"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Tom Hanks": {
        "age": 66,
        "movies": ["Forrest Gump", "Cast Away", "Saving Private Ryan"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Brad Pitt": {
        "age": 59,
        "movies": ["Fight Club", "Inglourious Basterds", "Once Upon a Time in Hollywood"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Denzel Washington": {
        "age": 68,
        "movies": ["Training Day", "Malcolm X", "Fences"],
        "Country of Origin": ["USA"],
    },
    "Will Smith": {
        "age": 54,
        "movies": ["The Pursuit of Happyness", "Men in Black", "Independence Day"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Hugh Jackman": {
        "age": 54,
        "movies": ["Logan", "The Greatest Showman", "Les Misérables"],
        "Country of Origin": ["Australia"],
    },
    "Matt Damon": {
        "age": 52,
        "movies": ["Good Will Hunting", "The Martian", "Jason Bourne"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Robert Downey Jr.": {
        "age": 58,
        "movies": ["Iron Man", "Sherlock Holmes", "Chaplin"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Jake Gyllenhaal": {
        "age": 42,
        "movies": ["Brokeback Mountain", "Nightcrawler", "Prisoners"],
        "Country of Origin": ["USA", "United States of America"],
    },
```

actors_database.py - C:\Users\kmaxe\AppData\Local\Programs\Python\Python312\Scripts\actors_database.py (3.12.0)

File Edit Format Run Options Window Help

```
    # Add more actors...
}
actresses = {
    "Meryl Streep": {
        "age": 73,
        "movies": ["The Devil Wears Prada", "Sophie's Choice", "The Iron Lady"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Jennifer Lawrence": {
        "age": 32,
        "movies": ["The Hunger Games", "Silver Linings Playbook", "Joy"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Emma Watson": {
        "age": 33,
        "movies": ["Harry Potter series", "Little Women", "Beauty and the Beast"],
        "Country of Origin": ["UK", "England", "United Kingdom"],
    },
    "Charlize Theron": {
        "age": 47,
        "movies": ["Monster", "Mad Max: Fury Road", "Atomic Blonde"],
        "Country of Origin": ["South Africa"],
    },
    "Scarlett Johansson": {
        "age": 38,
        "movies": ["Lost in Translation", "Avengers", "Marriage Story"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Natalie Portman": {
        "age": 42,
        "movies": ["Black Swan", "V for Vendetta", "Jackie"],
        "Country of Origin": ["Israel"],
    },
    "Anne Hathaway": {
        "age": 40,
        "movies": ["Les Misérables", "The Devil Wears Prada", "The Dark Knight Rises"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Viola Davis": {
        "age": 57,
        "movies": ["Fences", "The Help", "Doubt"],
        "Country of Origin": ["USA", "United States of America"],
    },
    "Cate Blanchett": {
        "age": 54,
        "movies": ["Blue Jasmine", "Carol", "The Aviator"],
        "Country of Origin": ["Australia"],
    },
    "Julianne Moore": {
        "age": 62,
```


actors_database.py - C:\Users\kmaxe\AppData\Local\Programs\Python\Python312\Scripts\actors_database.py (3.12.0)
File Edit Format Run Options Window Help
"Emily Blunt": {
"age": 40,
"movies": ["Edge of Tomorrow", "The Devil wears Prada", "The Quiet Place"],
"Country of Origin": ["UK", "England", "United Kingdom"],
},
"Rosamund Pike": {
"age": 44,
"movies": ["Gone Girl", "Jack Reacher", "Die Another Day"],
"Country of Origin": ["UK", "England", "United Kingdom"],
},
# Add more actresses...
}
def search_info(name):
if name in actors:
actor_info = actors[name]
return f"Name: {name}\nAge: {actor_info['age']}\nMovies: {', '.join(actor_info['movies'])}\nCo
elif name in actresses:
actress_info = actresses[name]
return f"Name: {name}\nAge: {actress_info['age']}\nMovies: {', '.join(actress_info['movies'])}
else:
return "Actor/Actress not found in the database."
def quiz():
questions = 5 # Set the number of questions for the quiz
all_actors = list(actors.keys())
all_actresses = list(actresses.keys())
total_names = all_actors + all_actresses
random.shuffle(total_names)
num_questions = min(questions, len(total_names)) # Ensure questions don't exceed available names
quiz_data = []
for i in range(num_questions):
name = total_names[i]
if name in actors:
category = "Actor"
info = actors[name]
else:
category = "Actress"
info = actresses[name]
question = {
"Name": name,
"Category": category,
"Age": info['age'],
"Movies": info['movies'],

Appendix B: Server code

quiz only server.py - C:/Users/kmaxe/AppData/Local/Programs/Python/Python312/Scripts/quiz only server.py (3.1...

File Edit Format Run Options Window Help

```
import socket
import pickle
import random
import json
from actors_database import search_info, quiz, actors, actresses

# server.py
def handle_client(client_socket):
    while True:
        message = client_socket.recv(1024)
        if not message:
            break

        try:
            data = pickle.loads(message)
            command = data.get("command")

            if command == "search_info":
                name = data.get("name")
                result = search_info(name)
                client_socket.send(pickle.dumps(result))
            elif command == "quiz":
                # Generate quiz questions
                quiz_questions = generate_quiz_questions()
                client_socket.send(pickle.dumps(quiz_questions))

                # Receive and evaluate user answers
                evaluate_quiz_answers(client_socket, quiz_questions)
            else:
                print("Invalid command")

        except Exception as e:
            print(f"Error processing client request: {e}")

def generate_quiz_questions():
    all_actors = list(actors.keys())
    all_actresses = list(actresses.keys())

    total_names = all_actors + all_actresses
    random.shuffle(total_names)

    quiz_questions = [{'name': name, 'category': 'Actor' if name in all_actors else 'Actress'}
    return quiz_questions

def evaluate_quiz_answers(client_socket, quiz_questions):
    for question in quiz_questions:
        # Send the question to the client
        client_socket.send(pickle.dumps(question))

        # Receive user answers from the client
        user_answers = pickle.loads(client_socket.recv(1024))

        # Evaluate user answers and send feedback
        feedback = evaluate_question(question, user_answers)
        client_socket.send(pickle.dumps(feedback))

    print("Quiz evaluation completed.")

def evaluate_question(question, user_answers):
    correct_age = str(actors[question['name']]['age']).lower() == user_answers['Age'].lower()
```

Ln: 26 Col: 0

```
def evaluate_question(question, user_answers):
    correct_age = str(actors[question['name']]['age']).lower() == user_answers['Age'].lower()
    correct_movie = user_answers['Movies'].title() in actors[question['name']]['movies']
    correct_country = user_answers['Country of Origin'].title() in actors[question['name']]['C

    feedback = {
        'age_correct': correct_age,
        'movie_correct': correct_movie,
        'country_correct': correct_country,
    }

    return feedback

    print("Quiz evaluation completed.")

if __name__ == "__main__":
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(("127.0.0.1", 5555))
    server.listen(5)

    print("Server listening on port 5555...")

    while True:
        client_socket, addr = server.accept()
        print(f"Accepted connection from {addr}")
        handle_client(client_socket)
```

Ln: 26 Col: 0

Appendix C: Client code

quiz only client.py - C:/Users/kmaxe/AppData/Local/Programs/Python/Python312/Scripts/quiz o...

File Edit Format Run Options Window Help

```
# client.py
import socket
import pickle

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(("127.0.0.1", 5555))

def take_quiz():
    quiz_data = {
        "command": "quiz"
    }
    client.send(pickle.dumps(quiz_data))

    for question_number in range(5):
        question = pickle.loads(client.recv(1024))
        print(f"\nQuestion {question_number + 1}:")
        print(f"Name: {question['Name']}")
        user_answers = {
            "Age": input("Enter the age: "),
            "Movies": input("Enter a movie from the list (comma-separated): "),
            "Country of Origin": input("Enter the country of origin: "),
        }
        client.send(pickle.dumps(user_answers))

        feedback = pickle.loads(client.recv(1024))
        print(f"Feedback for Question {question_number + 1}:")
        print(f"Age Correct: {feedback['age_correct']}")
        print(f"Movie Correct: {feedback['movie_correct']}")
        print(f"Country Correct: {feedback['country_correct']}")

    print("Quiz completed.")

def search_actor():
    name = input("Enter the name of the Actor/Actress: ")
    search_data = {
        "command": "search_info",
        "name": name
    }
    client.send(pickle.dumps(search_data))

    # Receive search result from the server
    search_result = pickle.loads(client.recv(1024))
    print(search_result)

def close_connection():
    client.close()

def main():
    while True:
        print("\nMenu:")
        print("1. Take Quiz")
        print("2. Search Actor/Actress")
        print("3. Close Connection")

        choice = input("Enter your choice (1/2/3): ")

        if choice == "1":
            take_quiz()
        elif choice == "2":
            search_actor()
        elif choice == "3":
```

```
            close_connection()
            break
        else:
            print("Invalid choice. Please enter 1, 2, or 3.")

if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        print(f"Error: {e}")
    finally:
        close_connection()
```