CHILD MANAGEMENT SYSTEM

import statistics

# In-memory storage for child records

child\_records = []

# CRUD Operations

def add\_child(name, age, guardian\_name, contact\_number):

child\_id = len(child\_records) + 1

child\_records.append({

"id": child\_id,

"name": name,

"age": age,

"guardian\_name": guardian\_name,

"contact\_number": contact\_number

})

print("Child record added successfully.")

def view\_children():

if not child\_records:

print("No child records found.")

for child in child\_records:

print(f"ID: {child['id']}, Name: {child['name']}, Age: {child['age']}, Guardian: {child['guardian\_name']}, Contact: {child['contact\_number']}")

def update\_child(child\_id, name, age, guardian\_name, contact\_number):

for child in child\_records:

if child["id"] == child\_id:

child["name"] = name

child["age"] = age

child

child["guardian\_name"] = guardian\_name

child["contact\_number"] = contact\_number

print("Child record updated successfully.")

return

print("Child ID not found.")

def delete\_child(child\_id):

global child\_records

child\_records = [child for child in child\_records if child["id"] != child\_id]

print("Child record deleted successfully.")

# Data Analysis and Reporting

def generate\_report():

if not child\_records:

print("No child records to analyze.")

return

total\_children = len(child\_records)

average\_age = statistics.mean(child["age"] for child in child\_records)

print(f"Total Children: {total\_children}")

print(f"Average Age: {average\_age:.2f}")

# Main Application

def main():

while True:

print("\nChild Management System")

print("1. Add Child Record")

print("2. View Child Records")

print("3. Update Child Record")

print("4. Delete Child Record")

print("5. Generate Report")

print("6. Exit")

choice = input("Enter your choice: ")

if choice == '1':

name = input("Enter child's name: ")

age = int(input("Enter child's age: "))

guardian\_name = input("Enter guardian's name: ")

contact\_number = input("Enter contact number: ")

add\_child(name, age, guardian\_name, contact\_number)

elif choice == '2':

view\_children()

elif choice == '3':

child\_id = int(input("Enter child ID to update: "))

name = input("Enter new child's name: ")

age = int(input("Enter new child's age: "))

guardian\_name = input("Enter new guardian's name: ")

contact\_number = input("Enter new contact number: ")

update\_child(child\_id, name, age, guardian\_name, contact\_number)

elif choice == '4':

child\_id = int(input("Enter child ID to delete: "))

delete\_child(child\_id)

elif choice == '5':

generate\_report()

elif choice == '6':

break

else:

print("Invalid choice. Please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

EXPLANATION

Overview

This Python program allows administrators to manage child records without using a database. Instead, it uses a list of dictionaries to store the records in memory. The program provides options to add, view, update, and delete child records. Additionally, it can analyze the data and generate a report on the records.

Components

1. In-Memory Storage

- `child\_records`: A list that stores child records, where each record is a dictionary.

2. CRUD Operations

- Functions to add, view, update, and delete child records.

3. Data Analysis and Reporting

- A function to generate a report on the child records.

4. Main Application

- A main loop that presents a menu to the user and handles their choices.

Detailed Explanation

In-Memory Storage

The `child\_records` list stores each child's information as a dictionary. This list is initialized empty and grows as records are added.

child\_records = []

CRUD Operations

**1. Add Child Record**

The `add\_child` function adds a new record to the `child\_records` list. It generates a unique `child\_id` based on the length of the list.

def add\_child(name, age, guardian\_name, contact\_number):

child\_id = len(child\_records) + 1

child\_records.append({

"id": child\_id,

"name": name,

"age": age,

"guardian\_name": guardian\_name,

"contact\_number": contact\_number

})

print("Child record added successfully.")

**2. View Child Records**

The `view\_children` function iterates over the `child\_records` list and prints each record. If the list is empty, it informs the user.

def view\_children():

if not child\_records:

print("No child records found.")

for child in child\_records:

print(f"ID: {child['id']}, Name: {child['name']}, Age: {child['age']}, Guardian: {child['guardian\_name']}, Contact: {child['contact\_number']}")

**3. Update Child Record**

The `update\_child` function updates an existing record by its `child\_id`. It searches the `child\_records` list for the record with the matching ID and updates its fields. If the ID is not found, it informs the user.

def update\_child(child\_id, name, age, guardian\_name, contact\_number):

for child in child\_records:

if child["id"] == child\_id:

child["name"] = name

child["age"] = age

child["guardian\_name"] = guardian\_name

child["contact\_number"] = contact\_number

print("Child record updated successfully.")

return

print("Child ID not found.")

**4. Delete Child Record**

The `delete\_child` function removes a record from the `child\_records` list by its `child\_id`. It uses a list comprehension to filter out the record with the specified ID.

def delete\_child(child\_id):

global child\_records

child\_records = [child for child in child\_records if child["id"] != child\_id]

print("Child record deleted successfully.")

**Data Analysis and Reporting**

The `generate\_report` function calculates and prints the total number of children and the average age. It uses the `statistics` module to compute the aveimport statistics

def generate\_report():

if not child\_records:

print("No child records to analyze.")

return

total\_children = len(child\_records)

average\_age = statistics.mean(child["age"] for child in child\_records)

print(f"Total Children: {total\_children}")

print(f"Average Age: {average\_age:.2f}")rage age.

**Main Application**

The `main` function runs an infinite loop to present a menu to the user. It takes input from the user to select an action and calls the corresponding function. The loop exits when the user chooses to exit.

def main():

while True:

print("\nChild Management System")

print("1. Add Child Record")

print("2. View Child Records")

print("3. Update Child Record")

print("4. Delete Child Record")

print("5. Generate Report")

print("6. Exit")

choice = input("Enter your choice: ")

if choice == '1':

name = input("Enter child's name: ")

age = int(input("Enter child's age: "))

guardian\_name = input("Enter guardian's name: ")

contact\_number = input("Enter contact number: ")

add\_child(name, age, guardian\_name, contact\_number)

elif choice == '2':

view\_children()

elif choice == '3':

child\_id = int(input("Enter child ID to update: "))

name = input("Enter new child's name: ")

age = int(input("Enter new child's age: "))

guardian\_name = input("Enter new guardian's name: ")

contact\_number = input("Enter new contact number: ")

update\_child(child\_id, name, age, guardian\_name, contact\_number)

elif choice == '4':

child\_id = int(input("Enter child ID to delete: "))

delete\_child(child\_id)

elif choice == '5':

generate\_report()

elif choice == '6':

break

else:

print("Invalid choice. Please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**How to Run the Program**

1. Save the Script:

- Save the entire script into a file named `child\_management\_system.py`.

2. Run the Program:

- Open a terminal or command prompt.

- Navigate to the directory where `child\_management\_system.py` is saved.

- Run the script:

python child\_management\_system.py

**Summary**

- The program manages child records using in-memory storage (a list of dictionaries).

- It provides functions to add, view, update, and delete records.

- It includes a function to analyze the data and generate a simple report.

- The main function presents a menu for user interaction and calls the appropriate functions based on the user's choice.