**Employee Management System project**

The Employee Management System (EMS) is a robust console-based application designed to facilitate efficient handling of employee records. This system streamlines human resource operations by automating core administrative tasks related to employee data management. With no reliance on external database systems, the EMS provides a simple yet effective solution for managing employee information directly within the application.

**Core Functionalities:**

1. **Add Employee**  
   Enables the addition of new employee records to the system.
   * **Details Captured:**
     + Employee ID (must be unique).
     + Name.
     + Age.
     + Gender.
     + Job Position.
     + Salary.
   * **Validation:**
     + The system enforces the uniqueness of Employee IDs to prevent duplication.
2. **Update Employee**  
   Provides the ability to modify existing employee records using their unique Employee ID.
   * **Submenu for Update Options:**
     + Users can selectively update specific fields such as Name, Age, Gender, Job Position, or Salary.
   * **Restriction:**
     + The Employee ID is immutable once assigned to an employee.
3. **Delete Employee**  
   Facilitates the removal of an employee's record based on their unique Employee ID.
4. **List Employees**  
   Displays all employee records in a well-formatted tabular structure, making it easy to review the organization's workforce.
5. **Exit System**  
   Allows users to gracefully terminate the application.

**User Interaction:**

* After successfully completing any operation (e.g., adding, updating, or deleting records), the system will prompt the user with the main menu options to perform further actions.
* A streamlined and intuitive menu-driven interface ensures a seamless user experience.

**Solution:**

# Employee Management System

#Globally a list has been created for employees to add

employees = []

# Add Employee

def add\_employee():

flag = True # for checking the uniqueness of the ID of the employee

print("Enter Employee Details:")

id = int(input("ID: "))

name = input("Name: ")

age = int(input("Age: "))

gender = input("Gender: ")

position = input("Position: ")

salary = float(input("Salary: "))

#Now Checking whether the ID provided by the user already exists or not

if not employees:

flag = True

else:

for emp in employees:

if id == emp['id']:

flag = False

break

if flag == True:

#If ID is unique then add the details

#Dictionary of employee with his details as key value pair has been added

employee = {'id': id, 'name': name, 'age': age, 'gender': gender, 'position': position, 'salary': salary}

#Now this detail is added in that globally declared list.

employees.append(employee)

print("Employee added successfully.")

else:

print('ID already exists!!')

# Update Employee

def update\_employee():

#Taking ID from the user on which update is to be performed.

print("Enter Employee ID:")

id = int(input("ID: "))

#Loop for finding the employee with the ID provided to update details

for employee in employees:

if employee['id'] == id: #condition for finding the employee

print('Which Information you want to update : ')

print('1. Name')

print('2. Age')

print('3. Gender')

print('4. Position')

print('5. Salary')

ch = int(input('Enter Your choice : '))

#Now checking for which information has to be updated

#On deciding, first line will show the current value of that column

#Second, new value will be asked from the user to update

if ch == 1:

#showing the current value of column name

print('Name : ', employee['name'])

#Taking the new value from the user to update

employee['name'] = input('Enter the new name :')

elif ch == 2:

print('Age : ', employee['age'])

employee['age'] = input('Enter the age :')

elif ch == 3:

print('Gender :', employee['gender'])

employee['gender'] = input('Enter the Gender : ')

elif ch == 4:

print('Name : ', employee['position'])

employee['position'] = input('Enter the position :')

elif ch == 5:

print('Salary : ', employee['salary'])

employee['salary'] = input('Enter the salary :')

print('Employee Details updated Successfully')

print()

#On successfully adding the employee

#return statement will return the control the while loop of options from

#add function to the while loop

return

#If employee with the ID provided is not found then no return

#statement will be executed and thus employee not found message

#Will be displayed to the user as written below.

print("Employee not found.")

# Delete Employee

def delete\_employee():

print("Enter Employee ID:")

id = int(input("ID: "))

for employee in employees:

if employee['id'] == id:

#On finding the employee with the concerned details

#Employee is removed from list employees using remove function of list

employees.remove(employee)

print("Employee deleted successfully.")

return

#here same logic for return is applied as explained in above add function.

print("Employee not found.")

# List Employees

def list\_employees():

if len(employees) == 0:

print("No employees found.")

return

#Printing in tabular format

print("ID\tName\tAge\tGender\tPosition\tSalary")

print()

for employee in employees:

print(f"{employee['id']}\t{employee['name']}\t{employee['age']}\t{employee['gender']}\

\t{employee['position']}\t{employee['salary']}")

# Main Loop

while True:

#Through this loop the execution of the program starts

#and this is the user view

print()

print("----Employee Management System-----")

print("1. Add Employee")

print("2. Update Employee")

print("3. Delete Employee")

print("4. List Employees")

print("5. Exit")

choice = int(input("Enter your choice: "))

if choice == 1:

add\_employee()

elif choice == 2:

update\_employee()

elif choice == 3:

delete\_employee()

elif choice == 4:

list\_employees()

elif choice == 5:

print('Program Exited!!')

break

else:

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

**Explanation:**

### ****Implementation Details of the Employee Management System****

The program is designed around a menu-driven interface, where the user interacts with various options provided in a continuous loop. The global list employees = [] serves as the central repository for storing employee data. Each employee's details are encapsulated in a dictionary, ensuring organized and structured data storage.

#### ****Functional Overview:****

1. **Global Repository:**
   * A global list employees = [] is initialized to maintain employee records throughout the program's lifecycle.
   * Each record in the list is represented as a dictionary, storing key details such as Employee ID, Name, Age, Gender, Job Position, and Salary.
2. **User Interaction Flow:**
   * The program begins with a while loop that serves as the main user interface.
   * Users are presented with options to perform various operations, and the control returns to this menu after each successful operation or message display.
3. **Core Functions:**
   * **add\_employee()**:
     + This function enables adding a new employee.
     + **Validation:**
       - It prompts the user for an Employee ID and checks for its uniqueness by searching the employees list.
       - If the ID already exists, the user is notified with an appropriate message, and the operation terminates.
     + **Insertion:**
       - If the ID is unique, additional details (Name, Age, Gender, Job Position, and Salary) are collected.
       - A dictionary containing the employee's details is appended to the employees list.
   * **update\_employee()**:
     + This function facilitates updating existing employee details.
     + **Search and Selection:**
       - It takes an Employee ID as input, searches the employees list, and retrieves the corresponding record.
       - If found, a submenu is displayed, allowing the user to select which attribute (e.g., Name, Age, etc.) they wish to modify.
     + **Value Update:**
       - The current value of the selected attribute is displayed, and the user is prompted for a new value, which replaces the old one in the dictionary.
   * **delete\_employee()**:
     + Removes an employee record based on the provided Employee ID.
     + **Validation:**
       - The function searches for the ID within the employees list. If found, the corresponding dictionary is removed; otherwise, a message indicating "Employee not found" is displayed.
   * **list\_employees()**:
     + Displays all employee records in a structured tabular format.
     + **Formatting:**
       - The function ensures data is presented clearly, with proper alignment of columns for attributes like ID, Name, and Salary.
4. **Return and Control Flow:**
   * **Use of return:**
     + The return statement is strategically employed to revert control to the main loop once an operation is completed successfully.
     + If an employee is not found (e.g., during an update or delete operation), the code bypasses the relevant if block, displays an appropriate message, and resumes interaction via the main menu.
5. **Error Handling:**
   * The program gracefully handles scenarios such as invalid Employee IDs, ensuring robust execution without crashing.
   * User prompts and feedback messages guide the interaction, ensuring clarity at every step.