**Employee Management System – Functionality #3**

Matt McPhail

CPT200: Fundamentals of Programming Languages

Professor Chintan Thakkar

April 28, 2025

**Purpose of Functionality:**

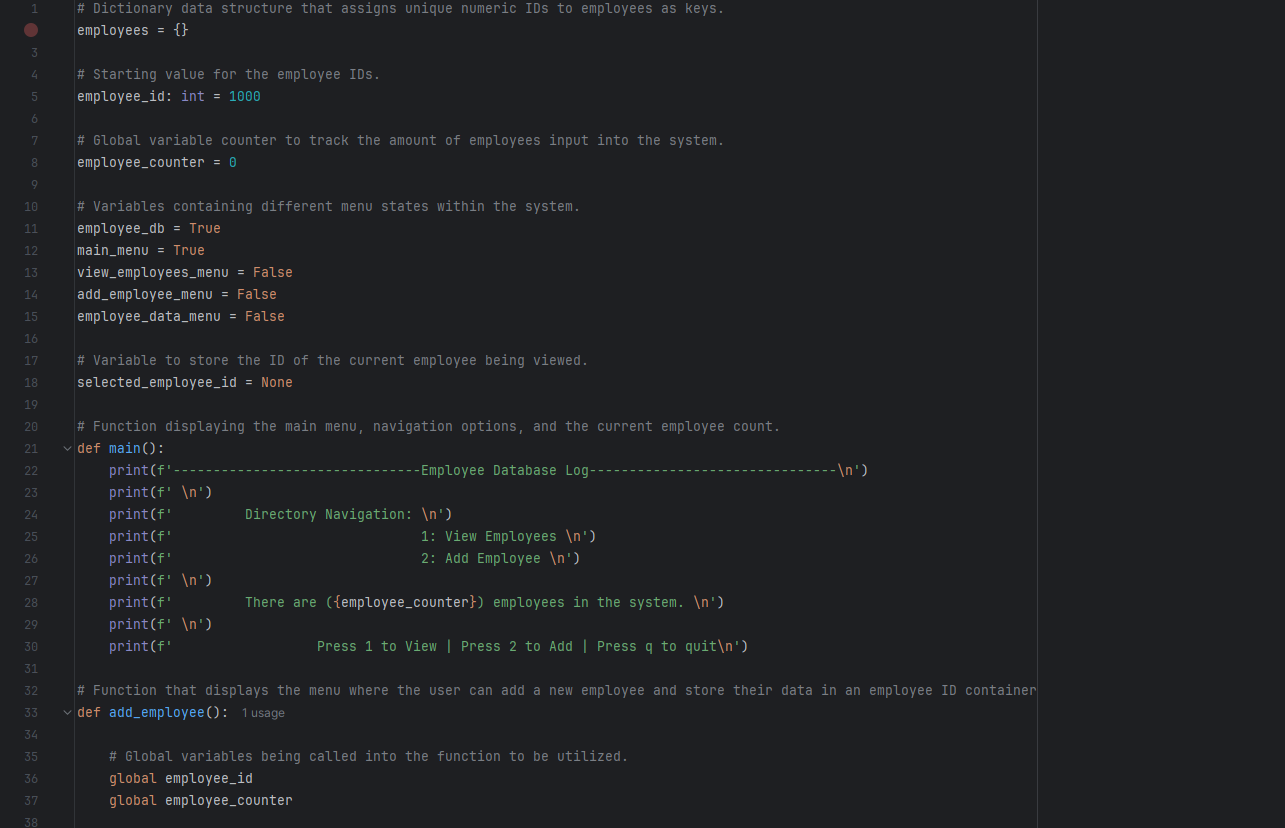
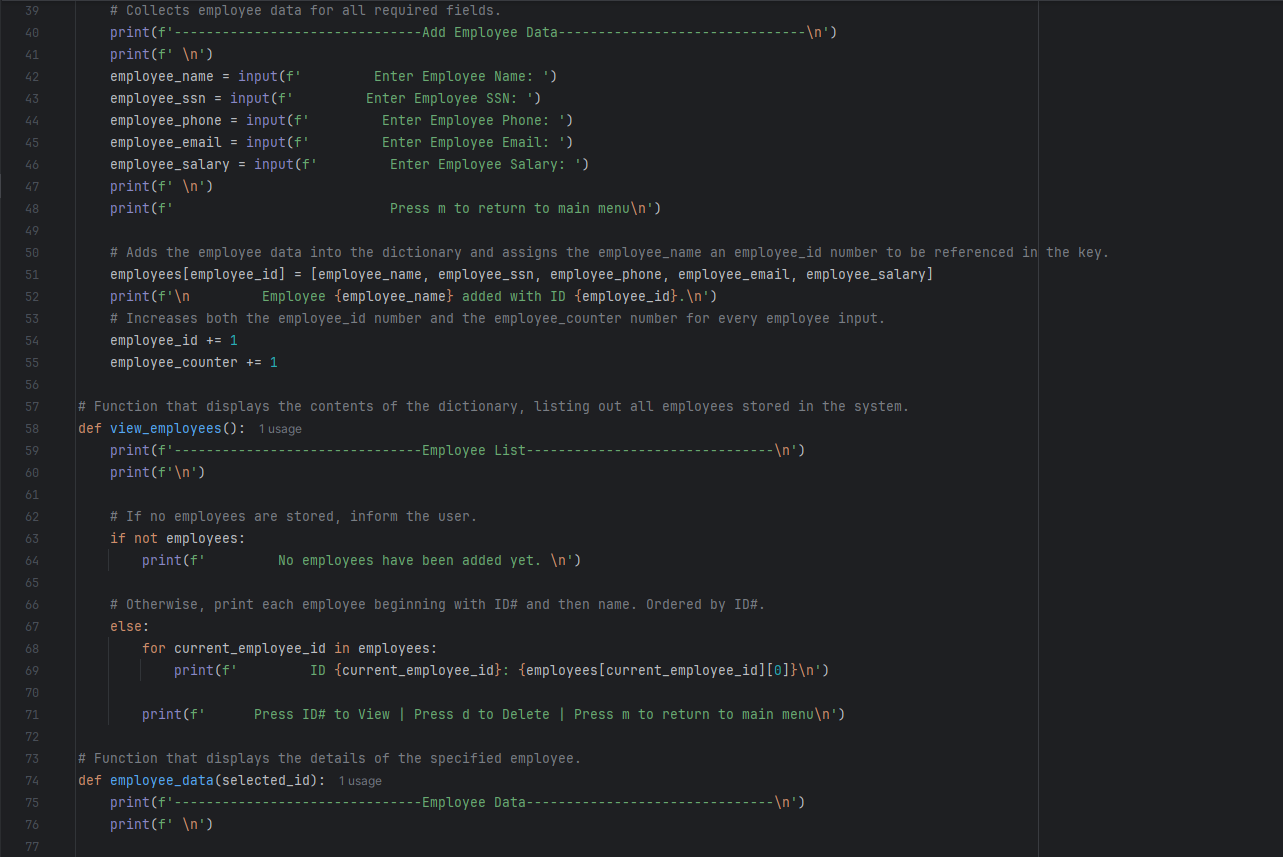
The purpose of this functionality was to build a basic employee management system where a user can add, view, and delete employee records without having to restart the program each time. This functionality needed to meet specific assignment criteria:

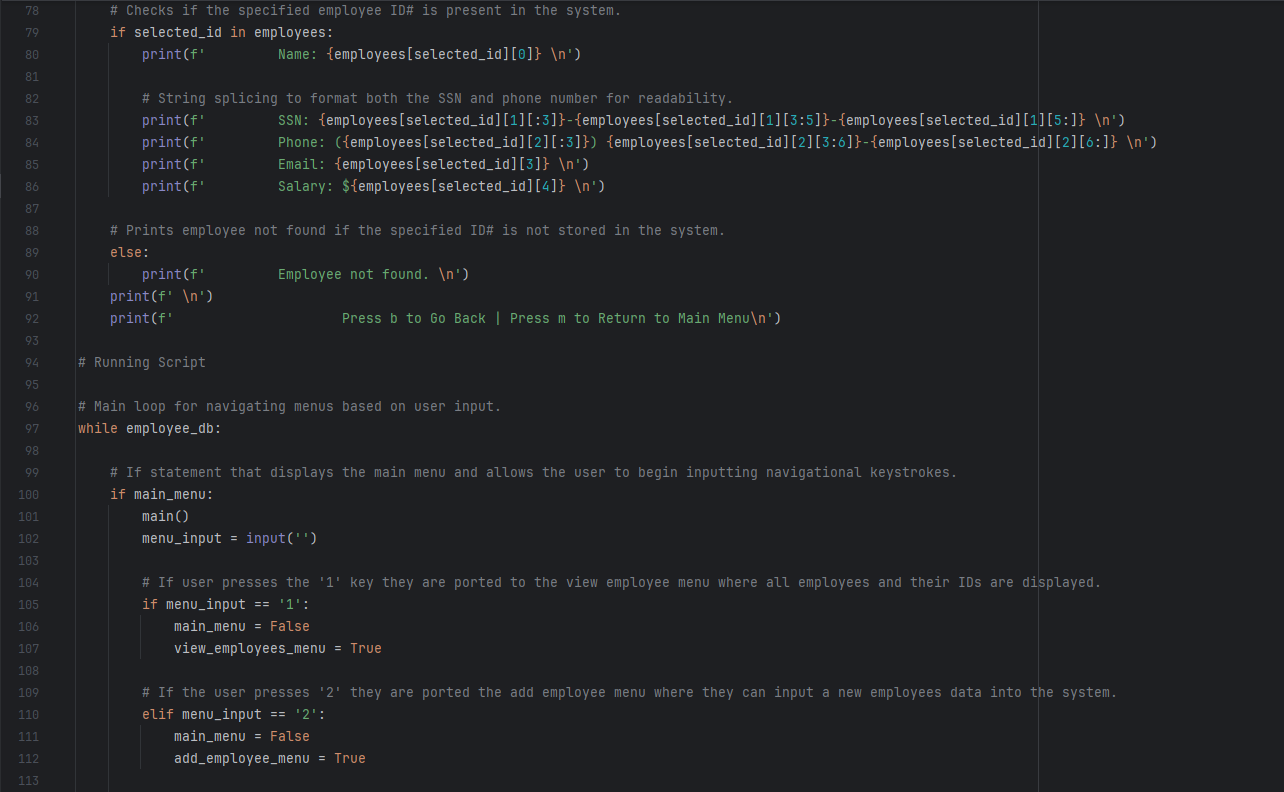
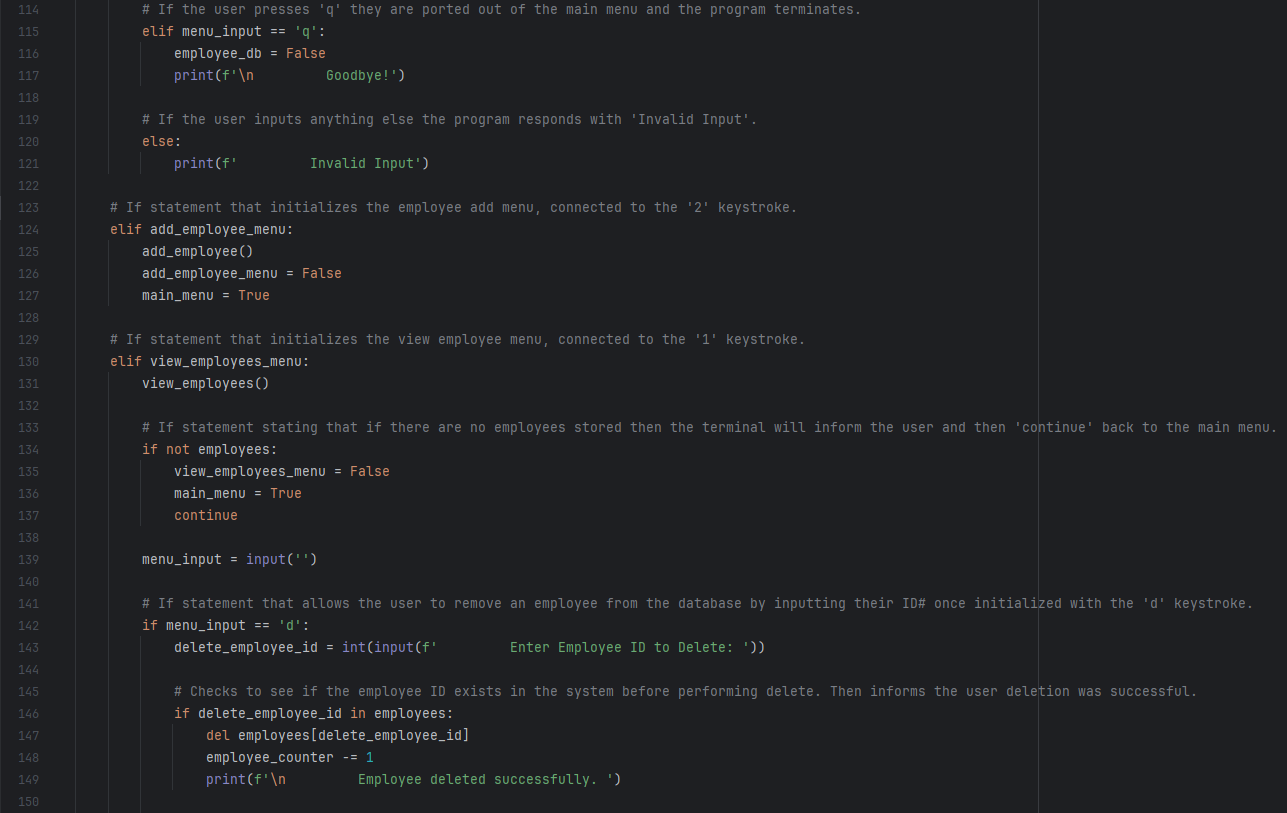
* Use a loop to keep the program running until the user exits.
* Separate Functionality 1 into two different functionalities.
* Use global variables to build a working counter for the employee system.
* Provide a brief explanation of what the functionality was meant to do.
* Attach the whole script.

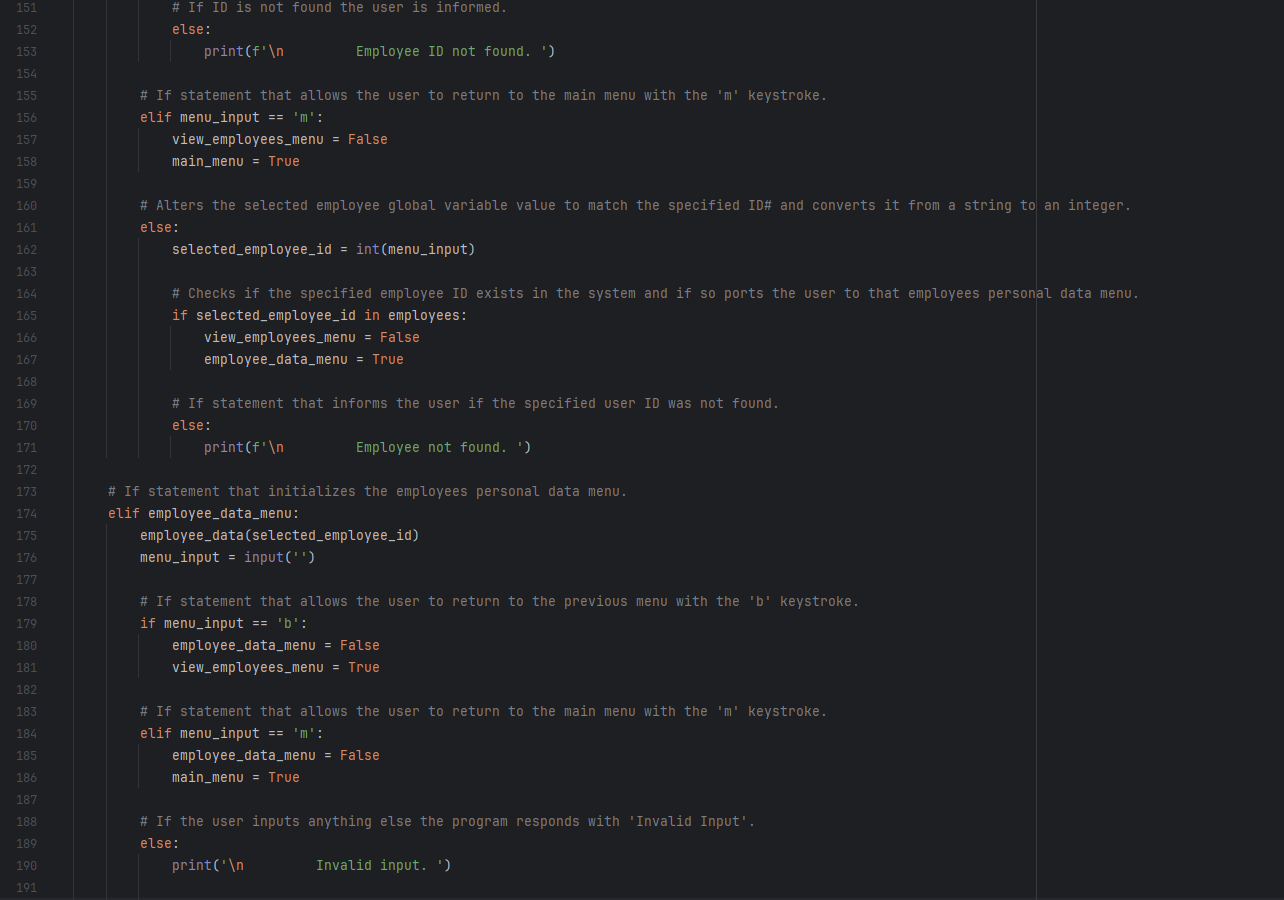
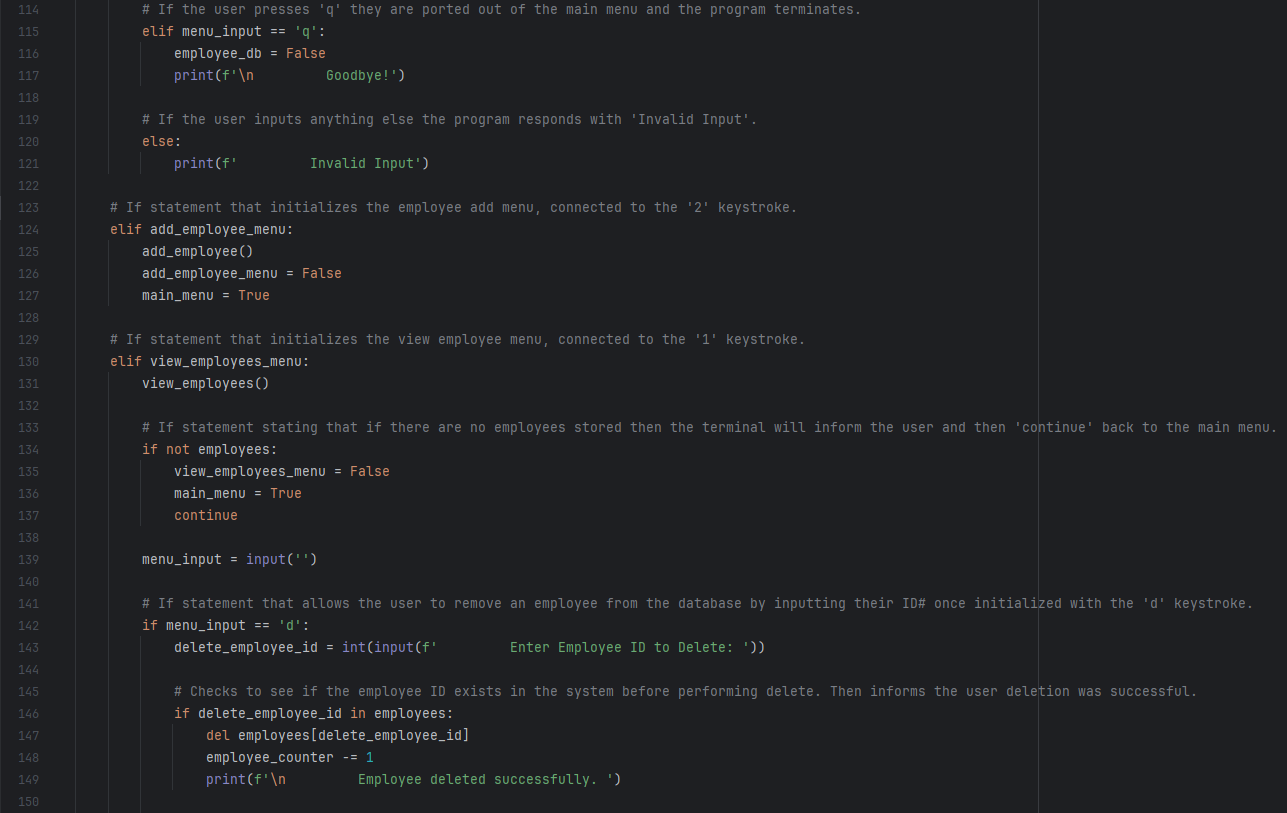
The goal of this functionality was to give the user a working but straightforward system for managing employee records and to demonstrate core skills like looping, using dictionaries, handling input, and using global variables for navigation. It ties everything we have been learning into a project that feels like building an actual tool, not just following instructions.

It also sets up the foundation for bigger projects later, like database systems or more advanced menu-driven programs.

**Screenshots of Script:**

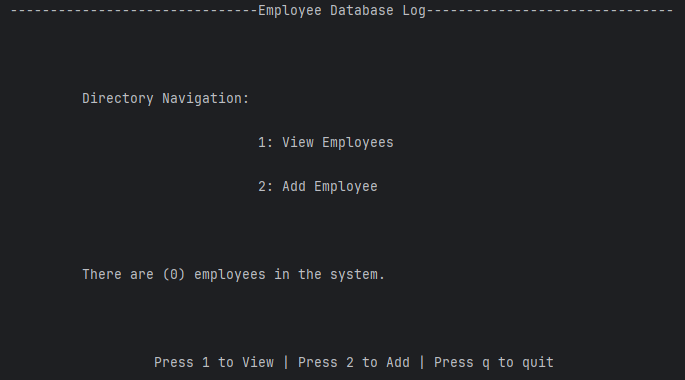
****

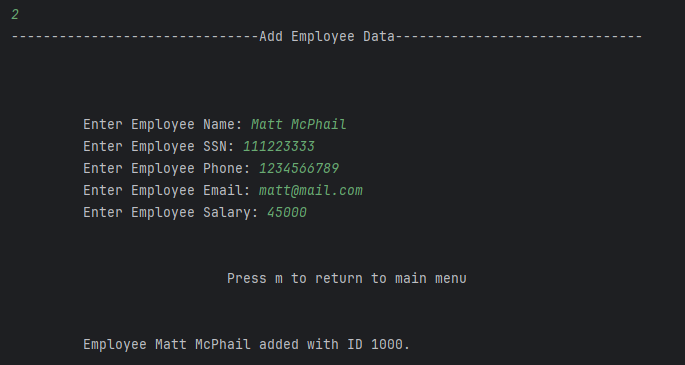
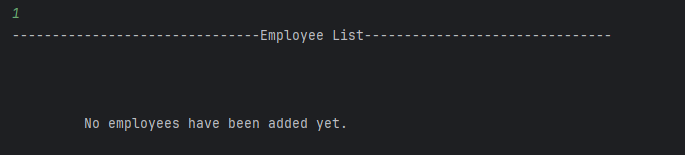
****

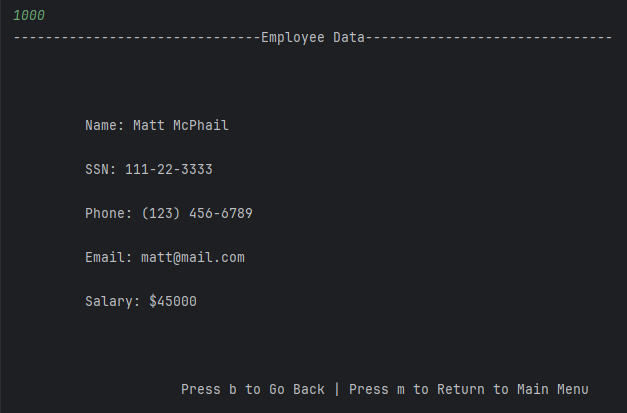
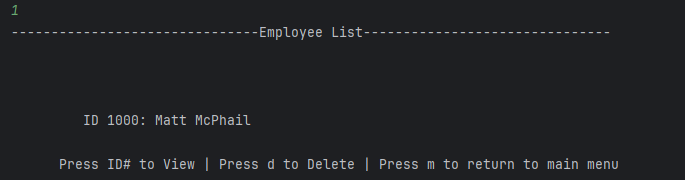
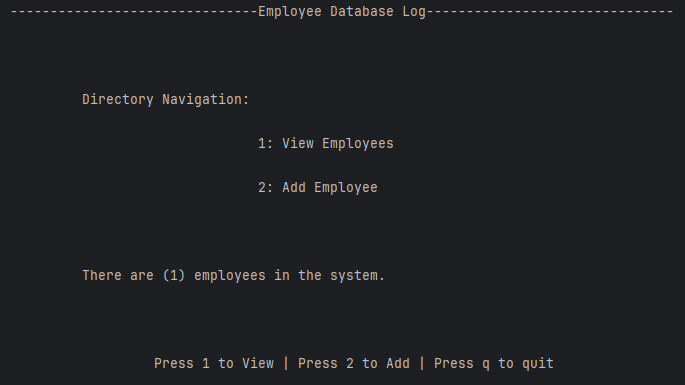


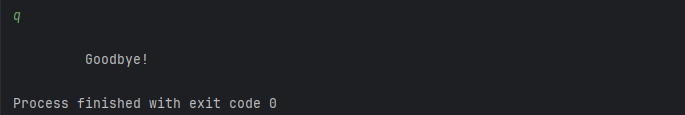
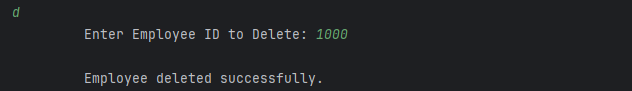
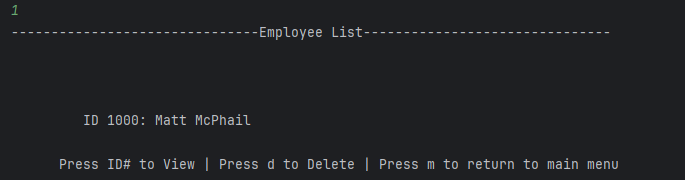
**Figure 1.** *Displays the full Python script used to build the Employee Management System. It includes all core functionalities such as adding employees, viewing employee lists, viewing individual employee details, and deleting employees, as well as the logic that governs the navigation between menus.*

**Screenshots of Functionality:**

****

****

****

****

**Figure 2.** *Displays the functionality of the Employee Management System while running. It shows how users can move between menus, add new employees, view a full list of employees, select individual employee records, delete employees by ID, and return to the main menu or exit the program. This figure demonstrates how the looping, Boolean flags, and user input work together to create a fully interactive system.*

**Functionality Breakdown:**

1. **How Looping Was Used to Keep the Program Running:**

One of the assignment requirements was to use a loop to make the program run constantly until the user quits. I set up a while employee\_db: loop as the program's backbone. As long as employee\_db stays True, the system keeps running. Inside the loop, I control which menu the user sees by flipping different Boolean flags like main\_menu, view\_employees\_menu, add\_employee\_menu, and employee\_data\_menu.

This setup lets the user stay inside the program, jump between different menus, and keep working without manually restarting the script. The loop ends when the user quits by pressing 'q,' which sets employee\_db = False.

1. **How I Split Functionality 1 Into Two Functionalities:**

Another requirement was to split up Functionality 1 into two different functionalities. Initially, managing employees was handled all at once, but I separated it into:

* **Viewing Employees:** Lists all employees by ID and lets users select one to view more details or delete them.
* **Adding Employees:** This lets users input new employee data like name, SSN, phone, email, and salary.

Splitting it up made the program easier to use and easier to maintain. It also cleaned up the main menu so users know exactly what they are doing at every step instead of getting overloaded with options.

1. **How I Used Global Variables to Track the Number of Employees:**

The assignment also required using global variables to build a working counter. I set up two globals at the top:

* employee\_id: Keeps track of which ID number to assign to the next employee.
* employee\_counter: Counts how many employees are stored at any given time.

I call these variables in the add\_employee() function with the global keyword. That way, when a new employee is added, the ID and counter update across the whole system without manually passing them into every function. It kept the data consistent and saved much extra work.

1. **Extras: Features Added Beyond Assignment Requirements:**

Even though it was not required in the assignment details, I added a few extras to make the program stronger and more user-friendly:

* **Formatted SSN and Phone Numbers:** I used string slicing to format the SSN and phone number outputs, so they look professional and readable instead of just printing raw digits.
* **Error Handling for Missing Employees:** If the user tries to view or delete an employee ID that does not exist, the program does not crash — it gives a clean "Employee not found" message.
* **Menu Navigation Flexibility:** I allowed users to bounce between menus using single key commands like 'm' (Main Menu) and 'b' (Back) instead of forcing full re-entries every time.

Adding these made the system feel more polished and closer to how real-world programs guide users and prevent crashes.