PYTHON GROUP PROJECT

Ayush Nitin Athare

MITU23BTITD003

Guide-Prof. ASHWINI JADHAV

Problem Statement- Develop a program to track daily tasks for users.

Theory :-

Class Definition (Task Management):

Data Structure:

In this project, we use a simple list to store tasks. Each task is represented as a string.

Lists are a fundamental data structure in Python and are suitable for storing collections of items.

User Interface:

The user interface is text-based and presented through the console/terminal.

The user is presented with a menu of options to interact with the to-do list.

Options include adding a task, removing a task, showing all tasks, and quitting the program.

Main Loop:

The program runs in an infinite loop (while True), allowing the user to perform multiple actions without restarting the program.

The loop continues until the user chooses to quit (choice == '4').

Menu and User Input:

The menu is displayed to the user at each iteration of the loop.

The user is prompted to enter their choice, and their input is captured using the input() function.

Functionality:

Add Task:

The user inputs a task, which is appended to the list of tasks.

A confirmation message is displayed to indicate that the task has been added successfully.

Remove Task:

If there are tasks in the list, the user is shown the current tasks along with their indices.

The user inputs the index of the task they want to remove, and the task is removed from the list.

Error handling is implemented to handle invalid indices.

Show All Tasks:

If there are tasks in the list, they are displayed one by one with their corresponding indices.

If the list is empty, a message indicating that there are no tasks is displayed.

Quit:

If the user chooses to quit, the program exits the loop and terminates.

Code Structure:

The functionality is implemented within the main() function.

he if __name__ == "__main__": block ensures that main() is executed when the script is run directly, allowing for modular and reusable code.get_daily_intake method retrieves the total daily water intake recorded by the tracker.

Main Function:

The main function serves as the entry point of the program. It creates an instance of the WaterTracker class and provides a menu-driven interface for users to interact with the water intake tracker.

The while loop continues to prompt the user for input until the user chooses to exit.

Input Validation:

The code includes input validation to handle cases where users enter invalid input, such as non-numeric values or negative numbers for water intake.

Exception Handling:

Exception handling using try-except blocks ensures that the program gracefully handles errors, such as invalid input, without crashing.

Overall, the code demonstrates the principles of encapsulation, abstraction, and modularity inherent in object-oriented programming, providing a structured approach to managing and tracking water intake.

This code creates a program to track daily water intake. It lets users add water, check their daily intake, or exit. It ensures users input valid data and keeps running until they choose to stop. It's a handy tool for staying hydrated!

Algorithm:-

=== To-Do List ===

1. Add Task

2. Remove Task

3. Show All Tasks

4. Quit

Code: -

```
def main():
    tasks = []

    while True:
        print("\n=== To-Do List ===")
        print("1. Add Task")
        print("2. Remove Task")
        print("3. Show All Tasks")
        print("4. Quit")

        choice = input("Enter your choice: ")

        if choice == '1':
            task = input("Enter the task: ")
            tasks.append(task)
            print("Task added successfully!")
        elif choice == '2':
            if tasks:
                print("Current Tasks:")
                for i, task in enumerate(tasks):
                    print(f"{i+1}. {task}")
                index = int(input("Enter the index of the task to remove:
")) - 1
```

```
                if 0 <= index < len(tasks):
                    removed_task = tasks.pop(index)
                    print(f"Task '{removed_task}' removed successfully!")
                else:
                    print("Invalid index!")
            else:
                print("No tasks to remove!")
        elif choice == '3':
            if tasks:
                print("Tasks:")
                for i, task in enumerate(tasks):
                    print(f"{i+1}. {task}")
            else:
                print("No tasks!")
        elif choice == '4':
            print("Exiting the program...")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

```
Process finished with exit code 0
```

Conclusion:-

Data Structures: Utilizing Python's built-in list to store and manage tasks efficiently.

User Interaction: Implementing a text-based user interface that allows users to interact with the to-do list through the console.

Control Structures: Employing loops (while loop) and conditional statements (if-elif-else) to control program flow.

Input Handling: Capturing user input using the input() function and validating it to ensure proper execution.