Technical Expectations

1. Purpose and Core Functionality

- The script implements a console-based Health Tracking App that allows users to record essential daily metrics: weight, calories consumed, and exercise details.
- It also enables users to view historically recorded data for each category, providing a simple health journal over time.
- The app is designed for ease of use with textual input/output and persistent data storage using plain text files.

---

2. Logging Mechanism (`log_data` function)

- A single reusable function `log_data(file_name, data)` handles appending data entries to respective files, ensuring all data is stored persistently.
- Opens files in append mode (`'a'`), so prior entries remain intact, and new logs are added sequentially.
- Adds a newline character after each entry for clear separation when viewing the logs.
- This modular design prevents code repetition for file handling across different input types, promoting code reusability and maintainability.

---

3. Recording Health Data

- Each type of data (weight, calories, exercise) has a dedicated function for input and logging:

  a. `record_weight()`
    - Prompts the user to enter the weight in kilograms.
    - Formats the input with the current date using `datetime.date.today()` for timestamping.
    - Stores the entry in the format `YYYY-MM-DD: <weight> kg` in `weight_log.txt`.
    - Confirms successful recording with a printed message.
- b. `record_calories()`

    - Prompts the user for daily calorie intake.
    - Logs the input with the current date in `calories_log.txt`.
    - Minor issue: appends "kg" which is a unit for weight — better to log as "kcal" or without a unit.
- c. `record_exercise()`
    - Collects details on exercise type and duration (minutes) separately.
    - Ideally should log as `YYYY-MM-DD: <exercise> for <duration> minutes`, but the existing code concatenates them imprecisely.
    - Stores data in `exercise_log.txt`.
    - Confirms entry via a user message.

---

4. Viewing Stored Data (`view_data`)

- Accepts user choice to select which data to view: weight, calories, or exercise.
- Uses a dictionary mapping choices (`'1'`, `'2'`, `'3'`) to corresponding filenames for streamlined access.
- Opens and reads the entire content of the selected log file, displaying all past entries.
- Implements exception handling for `FileNotFoundError` to gracefully inform users when no data has been recorded yet.
- Validates input choice to prevent invalid queries.

---

5. Interactive Command Loop (`main` function)

- Provides a menu-driven interface repeatedly displaying five options:
    1. Record weight
    2. Record calories consumed
    3. Record exercise
    4. View recorded data
    5. Exit program
- Processes user inputs with conditional branching to call appropriate functions based on choice.
- Ensures continued interaction until user explicitly chooses to exit.
- Handles unexpected choices by prompting users to try again, enhancing usability.

---

6. Technical and Design Analysis

- Modular design: Logical grouping of tasks into standalone functions supporting single responsibility.
- File I/O: Clean, context-managed reading and writing of data, ensuring files close automatically.
- Date time stamping: Use of Python's `datetime` ensures all entries are historical records with accurate dates.
- User interaction: Simple command-line interface with clear prompts and feedback messages.
- Error handling: Basic error management in viewing data files prevents crashes and informs the user.

---

7. Areas for Improvement to Match Professional Expectations

- Input validation: Currently, there is no check for valid numeric formats or reasonable ranges (e.g., negative weights or calories). Adding validation functions can improve robustness.
- Unit consistency: Fix calorie logging unit inconsistency ("kg" used instead of "kcal").
- Exercise log format: Clarify logged output by explicitly separating exercise type and duration.
- Code documentation: Adding function docstrings and inline comments would improve code readability and maintainability.
- Modularization: Refactor into classes or separate modules to enhance scalability and clean architecture.
- Testing: Integration of unit tests to verify functionality and input validation can solidify reliability.
- Version control & organization: Structuring with directories for source code, data, and tests would better support project versioning and collaboration.

---

8. Summary: How This Code Fulfills the Task

- The code successfully implements the fundamental features of a health tracking application:
    - Capturing user input for weight, calories, and exercise.

    - Storing the inputs persistently with dates for historical tracking.
    - Retrieving and displaying stored data on demand.
- It showcases key programming concepts such as input/output processing, file handling, modular function design, and error handling.
- This fulfills the educational task of writing a basic, working health tracking app while leaving room for further sophistication and professional best practices.
