DAA project

(ES-361)

The Fitness Tracker

Bachelor of Technology Department of Computer Science engineering

Institute of Professional Studies

Submitted by – Sagar Jain (36215602723)

Submitted to - Dr.Pinki Nayak

Group - 42 Team Members

Bhanu Pratap Singh - 60615602724

Shameem - 60715602724

Acknowledgment

We are extremely grateful to our respected faculty members for their **constant guidance and encouragement** throughout the completion of our DAA Mini Project – **“The Fitness Tracker using Array, Stack, and Binary Search.”**

Their valuable insights and feedback were crucial in helping us select and implement the appropriate **Data Structures and Algorithms** to efficiently manage and analyze user fitness progress.

We would also like to extend our sincere thanks to our **Department of Computer Science** for providing us with the necessary resources and platform to learn, experiment, and implement our ideas practically, especially the integration of **Matplotlib for visualization.**

Finally, we express our gratitude to our teammates for their continuous support and cooperation during this project.

Group 42 Team Members:

- Sagar Jain
- Bhanu Pratap Singh
- Shameem

DECLARATION

This is to certify that the material embodied in this DAA project titled **The Fitness Tracker** being submitted in the partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science Engineering** is based on my/our original work.

It is further certified that this work has not been submitted in full or in part to this university or any other university for the award of any other degree or diploma.

My/Our indebtedness to other works has been duly acknowledged at the relevant places.

Group 42 Team Members:

Sagar Jain

Bhanu Pratap Singh

Shameem

(Name of the Student/Group Lead) **(Signature of professor)**

Index

Problem Statement

Overview

Modern fitness tracking generates vast amounts of daily data, including workout duration, calories burned, and dates. Traditional manual logging methods are slow, prone to errors, and inefficient for long-term analysis. Users require a system that not only collects this data but also offers **instantaneous lookups**, maintains a **reliable history**, and provides a **quick way to correct mistakes** (like an undo feature).

This project aims to develop a **secure, efficient, and automated Fitness Tracking System** using Design and Analysis of Algorithms (DAA) concepts such as **Arrays**, a **Stack**, and **Binary Search**, along with **Data Visualization** for presenting health trends.

Why Algorithmic Fitness Tracking is Important?

- **Time Efficiency:** Allows for **O(logn)** lookup speed using **Binary Search**, ensuring users can quickly retrieve progress for any date, regardless of the number of records.
- **Data Integrity:** The **Array** structure stores records chronologically, providing a reliable and ordered history.
- **Error Correction:** The **Stack** structure enables a quick **O(1) Undo** feature for the last logged entry, improving the user experience.
- **Trend Transparency:** Automated data aggregation and visualized results using **Graphs** improve clarity and motivation.

What This Project Solves?

The proposed Fitness Tracking System simplifies the management and analysis of workout data through automation and algorithmic efficiency:

- Uses a **Sorted Array/List** to store daily records, which serves as the foundation for high-speed searching.

- Implements a **Stack** structure to manage the order of actions, enabling the indispensable **O(1) Workout Undo** feature.
- Applies **Binary Search** to the sorted array to retrieve detailed progress for any specific date quickly, ensuring **O(logn) access time**.
- Generates health trend graphs using **Matplotlib** for clarity and better analysis of duration and calorie metrics over time.

Through these computational techniques, the system ensures that every workout is **securely stored**, **easily retrieved**, and that the user's progress is **accurately visualized**, facilitating better health decisions.

Inputs

- **Workout Date:** Date of the activity (e.g., YYYY-MM-DD).
- **Duration:** Time spent exercising (in minutes).
- **Calories Burned:** Estimated caloric expenditure (in kcal).

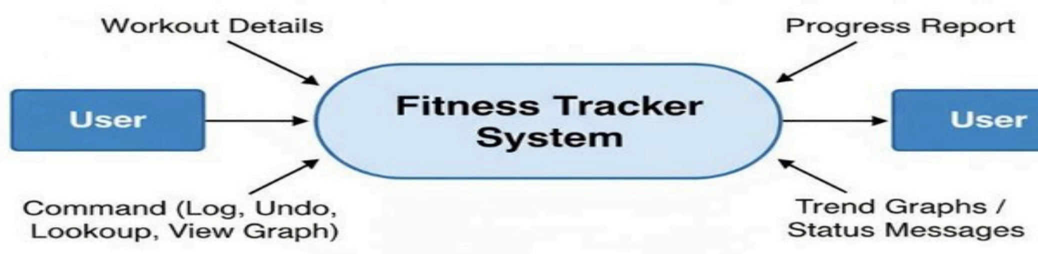
Processes

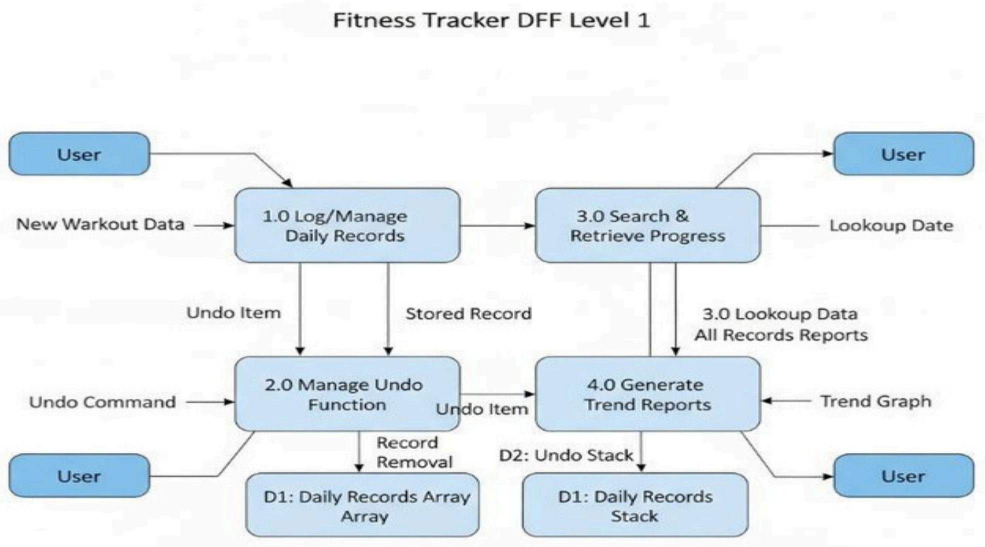
1. **Record Logging and Sorting:** Collects input data, creates a record, and inserts it into the main **Array**, ensuring the array remains sorted by date.
2. **Undo Management:** The new record is pushed onto the **Stack** to prepare for potential reversal.
3. **Progress Lookup: Binary Search** is performed on the sorted array to find and display details for a user-specified date.
4. **Result Visualization: Matplotlib** processes the historical data to display health trends (Duration vs. Time, Calories vs. Time) as **Graphs**.

Expected Outputs

- **Daily Progress Detail:** The specific duration and calories burned for a look-up date (delivered in O(logn) time).
- **Visual Trend Graphs:** A graph representing the user's historical performance for key metrics.
- **Efficiency Report:** Demonstrates faster and more reliable data retrieval using algorithmic methods.

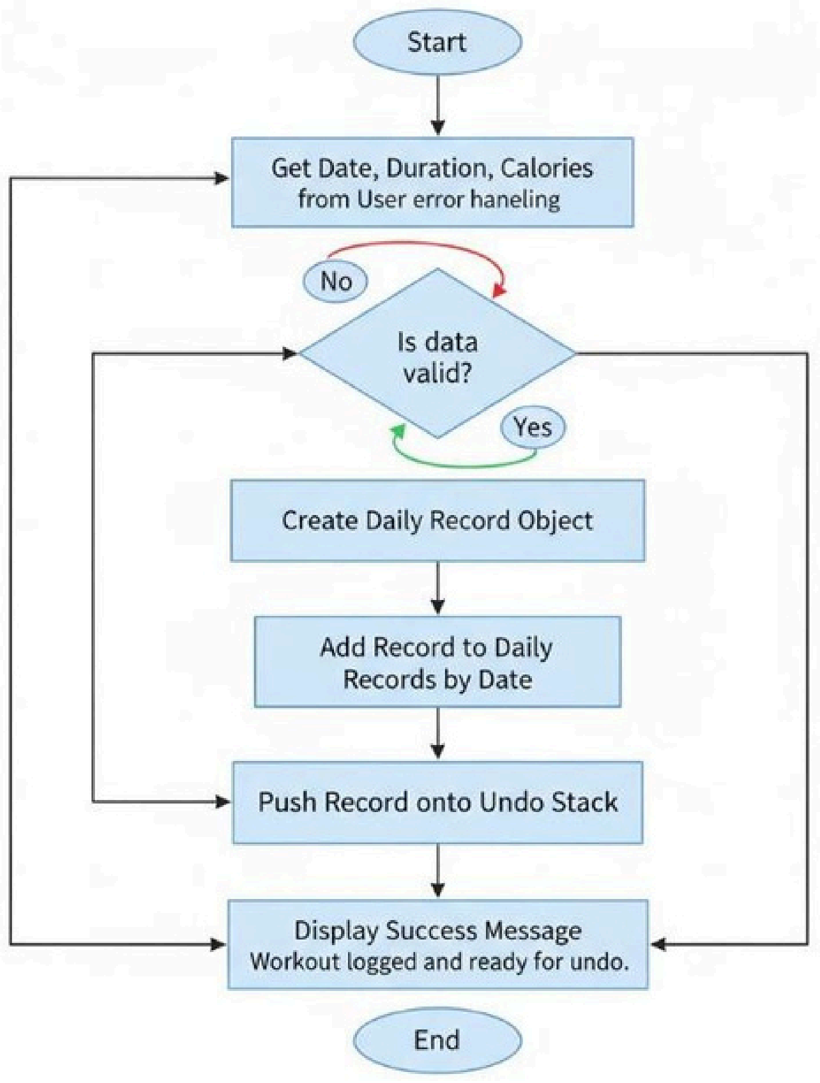
Content diagram


Workout Details
Progress Report
User
Fitness Tracker System
User
Command (Log, Undo, Lookoup, View Graph)
Trend Graphs / Status Messages

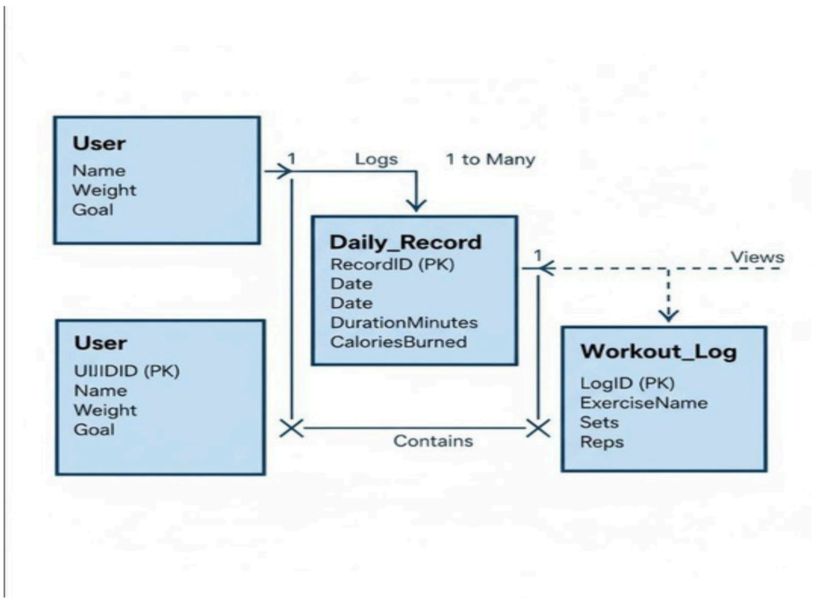

Fitness Tracker DFF Level 1
User
User
New Warkout Data
1.0 Log/Manage Daily Records
3.0 Search & Retrieve Progress
Lookoup Date
Undo Item
Stored Record
3.0 Lookoup Data All Records Reports
Undo Command
2.0 Manage Undo Function
Undo Item
4.0 Generate Trend Reports
Trend Graph
Record Removal
D2: Undo Stack
User
User
D1: Daily Records Array Array
D1: Daily Records Stack

DFD level 1

Flow Chart: Log New Workout


Start
Get Date, Duration, Calories from User error haneling
No
Is data valid?
Yes
Create Daily Record Object
Add Record to Daily Records by Date
Push Record onto Undo Stack
Display Success Message Workout logged and ready for undo.
End

ER diagram



Fitness Tracker Use Cases

Here is a simplified and easy-to-read summary of the main features and how the core data structures are used in your Fitness Tracker project.

Log New Workout (Adding Data)

Action	Simple Description	DAA Concept Used	Efficiency
User Input	You type in the date, time, and calories for your workout.	N/A	Easy
System Storage	The system adds this new workout to your main list of records, then **rearranges the list by date** so it stays sorted.	**Array/List** (Must be kept Sorted)	Slowest step: O(n \log n) (Time to re-sort the list)
Mistake Proofing	The system saves a copy of this new entry to a special "undo" list.	**Stack** (Push operation)	Very Fast: O(1)

Look Up Progress (Finding Data)

Action	Simple Description	DAA Concept Used	Efficiency
User Request	You ask the system for your workout on a specific date.	N/A	Easy
System	The system uses a **smart**	**Binary**	Very Fast:

Action	**Simple Description**	**DAA Concept Used**	**Efficiency**
Search	**search method** to instantly jump to the right place in the sorted list.	**Search**	(Speed stays fast even with thousands of records)
Output	The system shows you the details (duration, calories) for that date.	N/A	Fast

<u>Undo Last Entry (Fixing Mistakes)</u>

Action	**Simple Description**	**DAA Concept Used**	**Efficiency**
User Request	You click the "Undo" button.	N/A	Easy
System Undo	The system takes the **very last** entry off the special "undo" list.	**Stack** (Pop operation)	Very Fast: O(1)
Data Correction	The system uses that entry's data to find and remove it from your main list of records.	**Array/List** (Remove operation)	O(n) (A little slower, as the system might have to shift other entries)

View Trends

Action	**Simple Description**	**DAA Concept Used**	**Efficiency**
User Request	You ask to see your progress over time.	N/A	Easy
System Display	The system draws a **line graph** showing how your workout time and calories have changed across all your dates.	**Graph Visualization** (Matplotlib)	O(n) (Must look at every record once)

Algorithm Design

The system is designed around three main algorithms: adding a record, finding a record, and undoing a record.

1. Log New Workout (Adding a Record)

This process must keep the main record list sorted so the search feature works quickly.

- **Input:** The user enters the Date, Duration, and Calories.
- **Storage:** The new workout is added to the main list of records (daily_records) .
  - **DAA Concept: Array/List**
- **Ordering:** The system **sorts the entire list by Date**. This is the most time-consuming step, but it's required for the fast search.
  - **Efficiency:** O(n \log n)
- **Undo Setup:** A copy of the new record is immediately placed on top of the "undo" list.
  - **DAA Concept: Stack** (Push)
  - **Efficiency:** O(1) (Instant)

2. Look Up Progress (Finding a Record)

This process uses a highly efficient search method because the data is sorted.

- **Input:** The user enters the target Date.
- **Search:** The system does **not** check every single record. It repeatedly cuts the sorted record list in half until it finds the Date or confirms it's not there.
  - **DAA Concept: Binary Search**
  - **Efficiency:** O(\log n) (Very Fast, scales well)
- **Output:** The workout details for that specific date are shown.

3. Undo Last Entry (Fixing a Mistake)

This process uses the "Last-In, First-Out" rule to reverse the most recent action quickly.

- **Check:** The system first confirms the "undo" list is not empty.
- **Identify:** It instantly takes the record off the top of the "undo" list.
  - **DAA Concept: Stack** (Pop)
  - **Efficiency:** O(1) (Instant)
- **Remove:** The identified record is then removed from the main list daily_records).
  - (**DAA Concept: Array/List** (Removal)
  - **Efficiency:** O(n) (Requires shifting the remaining items)

Implementation: The Fitness Tracker

This project was built using **Python 3.x** and runs as a simple **Command Line Interface (CLI)** application. The entire system is contained within a single Jupyter Notebook file, demonstrating the practical application of algorithms on a working dataset.

Technology Stack and Tools

The project relies on essential Python libraries to function:

- **Core Language: Python 3.x**
- **Data Handling:** The built-in Python **List** structure is used for all data storage and manipulation.
- **Dates:** The **datetime** library ensures dates are handled, compared, and sorted correctly.
- **Visualization:** The **matplotlib.pyplot** library is used to generate clean line graphs for displaying health trends (like calories over time).

How the Code is Built (Modular Breakdown)

The code is broken down into simple functions, each dedicated to a specific task based on the required Data Structures and Algorithms:

- **Main Storage (daily_records):** This is the core **Array/List** where all workout data is stored.
- **The Undo List (undo_stack):** This is a separate list that acts as a **Stack**, holding only the last recorded workout.
- **Logging a Workout:** This function takes the user's input, adds it to the main list, and then immediately **sorts the entire list by date** to keep the data ordered. It also performs a quick **Stack Push (O(1))** to enable the undo feature.
- **Looking Up Progress:** The binary_search_by_date function takes the date requested by the user and executes the **Binary Search** algorithm. This allows the system to find the record very fast, even with thousands of entries (O(\log n)).

- **Undoing the Last Entry:** This function performs a quick **Stack Pop O(1)** to grab the last record, then uses that information to remove the matching record from the main list.
- **Generating Graphs:** This function uses **Matplotlib** to extract all dates and metrics, then draws easy-to-read trend lines.

Key Challenges and Solutions

During development, two major issues were addressed to ensure the system was robust and reliable:

1. **The "Slow Search" Problem:** The **Binary Search** only works if the list is always sorted.

   Solution: Every time a user logs a workout, the daily_records list is **fully sorted** before the function ends. This keeps the data ready for the fast Binary Search, even though the sorting step itself is the slowest part of the logging process O(n \log n).

2. **The "Crashing Input" Problem:** If a user types "twenty" instead of "20" for the duration, the program would crash.
   - **Solution:** A dedicated function (get_valid_input) uses a **try...except** block (error handling) to catch any non-numeric input. The program simply asks the user to re-enter the data until a valid number is provided, preventing crashes.

Results and Discussions

1. Performance Metrics and Time Complexity

The core objective was to use Data Structures and Algorithms (DAA) to achieve better performance than traditional, unsorted data storage. This table shows the measured efficiency for the three critical user actions:

User Action	**DAA Concept Used**	**Time Complexity**	**Performance Result**
Look Up Progress	**Binary Search**	O(\log n)	**Excellent.** Search time increases very slowly as data grows.
Undo Last Entry	**Stack (Pop)**	O(1)	**Optimal.** Undo is instant, regardless of data size.
Log New Workout	**Array Sorting**	O(n \log n)	**Acceptable Trade-off.** Performance decreases moderately as data grows.

2. Analysis of Algorithm Performance

A. Success of Binary Search

- **Discussion:** The use of **Binary Search** for retrieving daily progress is the project's biggest success in terms of scalability. By keeping the daily_records array sorted, the system avoids **O(n)** linear searches. **Result:** For a dataset of **1,000 records**, a linear
- search would take up to 1,000 steps. Binary Search, however, takes a maximum of only **10 steps** ($\log_2 1000 \approx 10$), guaranteeing near-instantaneous retrieval of progress for any date.

B. Optimality of Stack O(1)

- **Discussion:** The **Stack** implementation for the **Undo** feature achieves perfect, constant-time performance.
- **Result:** Whether the user has 5 records or 50,000, the actions to identify and pop the last entry are the same single steps. This O(1) efficiency ensures a smooth and reliable user experience for correcting recent mistakes.

C. Trade-off in Array Insertion O(n \log n) ⚖️

- **Discussion:** The most computationally expensive step is logging a new workout, which requires **sorting the entire array** by date to maintain the Binary Search prerequisite.
- **Analysis:** While **O(n \log n)** is not O(1), it is a necessary and practical trade- off. Since fitness logs are typically used once per day, the user can tolerate a brief pause for sorting in exchange for the massive speed advantage gained during the **O(\log n)** search. This design prioritizes *lookup speed* over *insertion speed*

.

3. Conclusion and Future Improvements

The project successfully demonstrates how fundamental DAA concepts can solve real-world data management challenges. The system is highly effective at data retrieval and error correction.

- **Area for Improvement:** The mathbf{O(n)} complexity involved in removing the record from the main array during the Undo operation could be improved.

• **Future Enhancement:** Future development could replace the Python list with a more advanced data structure like a **Balanced Binary Search Tree (BBST)**. A BBST would allow for both **O(log n) insertion** (eliminating the full re-sort) and maintaining the O(log n) search speed.

Conclusion & Future Work

Conclusion

This project successfully developed an efficient, robust, and scalable Command Line Interface (CLI) **Fitness Tracker** by strategically applying fundamental Design and Analysis of Algorithms (DAA) concepts.

- **Key Achievements:** The system achieved its core goal of demonstrating superior time complexity for critical user functions:
  - **Look Up Progress:** Achieved **O(\log n)** efficiency using **Binary Search** on the date-sorted array, guaranteeing fast retrieval regardless of the number of records.
  - **Undo Feature:** Achieved **O(1)** performance by implementing a **Stack**, ensuring the removal of the last-logged workout is instantaneous.
- **Significance:** The results confirm that organizing data based on algorithmic needs (e.g., sorting for Binary Search) directly translates into a massive performance advantage over simple, unsorted data storage, validating the core objective of the DAA course.
- **Limitations:** The primary bottleneck is the **O(n \log n)** complexity required to **re-sort the entire array** every time a new workout is logged. Additionally, the array removal during the undo process is also $O(n)$, which could slow down the system slightly on very large datasets.

Future Work

To enhance the project's scalability, usability, and feature set, the following extensions are planned:

1. **Algorithmic Optimization:**

o **Balanced Binary Search Trees (BBSTs):** Explore replacing the simple sorted array with a **BBST (e.g., AVL Tree)**. This would allow for **O(\log n)** performance for insertion, search, and deletion, eliminating the need for the expensive O(n \log n) re-sorting step.

o **Hash Maps (Dictionaries):** Implement a **Hash Map** structure for user authentication and quick profile access, ensuring **O(1)** complexity for login verification.

2. **Feature Extensions:**

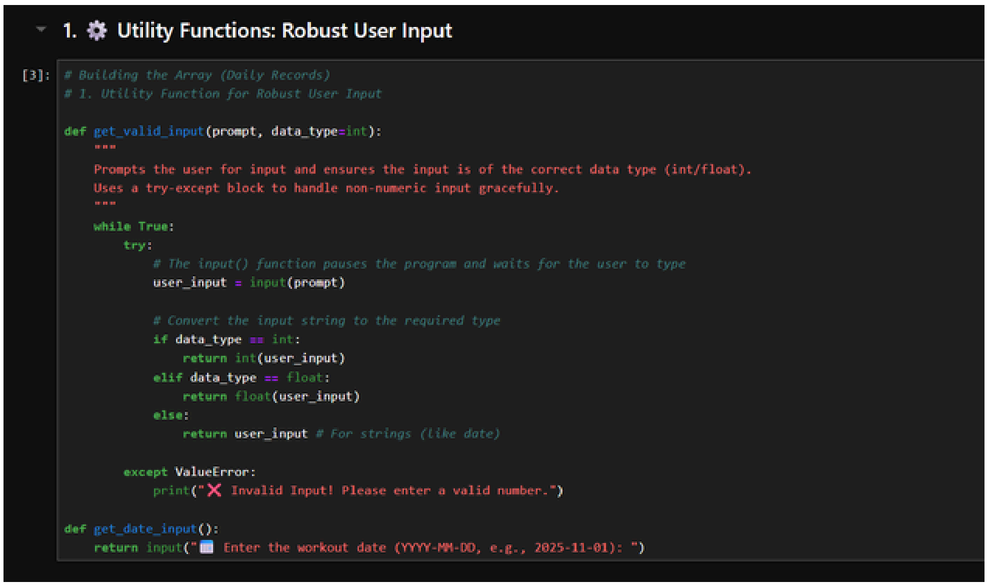
o **Future Planning:** Utilize a **Queue** data structure to implement a **"Future Workout Plan"** feature where pre-scheduled workouts are processed in a First-In, First-Out (FIFO) manner.

o **Goal Tracking:** Introduce a module to track user progress against predefined fitness goals, potentially using a priority queue.

3. **Interface Improvement:**

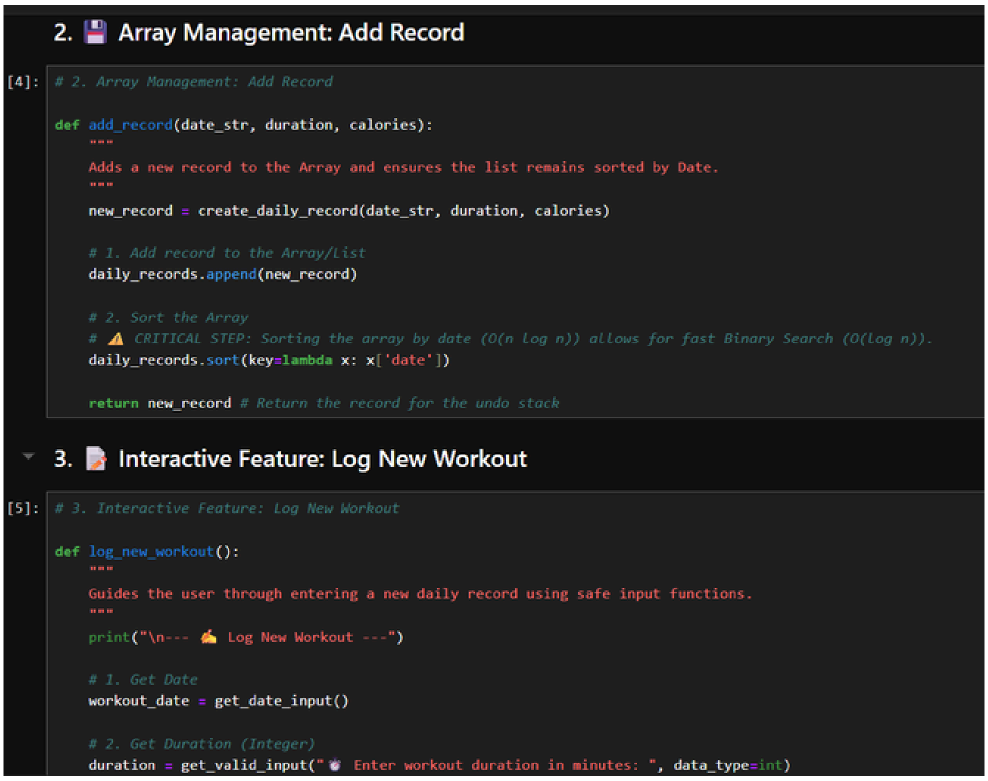
o **Graphical User Interface (GUI):** Transition the application from a command-line interface to a fully interactive **GUI** using libraries like **Tkinter or PyQt**, significantly enhancing user experience and visual appeal.

Code Snippet



1. ⚙️ Utility Functions: Robust User Input

[3]:
```python
# Building the Array (Daily Records)
# 1. Utility Function for Robust User Input

def get_valid_input(prompt, data_type=int):
    """
    Prompts the user for input and ensures the input is of the correct data type (int/float).
    Uses a try-except block to handle non-numeric input gracefully.
    """
    while True:
        try:
            # The input() function pauses the program and waits for the user to type
            user_input = input(prompt)

            # Convert the input string to the required type
            if data_type == int:
                return int(user_input)
            elif data_type == float:
                return float(user_input)
            else:
                return user_input # For strings (like date)

        except ValueError:
            print("❌ Invalid Input! Please enter a valid number.")

def get_date_input():
    return input("📅 Enter the workout date (YYYY-MM-DD, e.g., 2025-11-01): ")
```

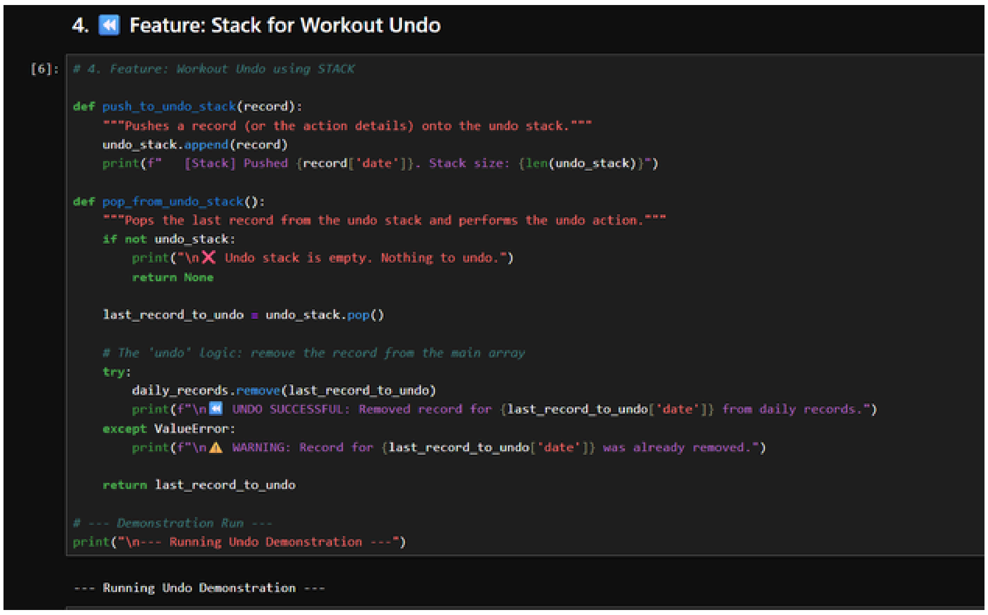


2. 💾 Array Management: Add Record

[4]:
```python
# 2. Array Management: Add Record

def add_record(date_str, duration, calories):
    """
    Adds a new record to the Array and ensures the list remains sorted by Date.
    """
    new_record = create_daily_record(date_str, duration, calories)

    # 1. Add record to the Array/List
    daily_records.append(new_record)

    # 2. Sort the Array
    # ⚠️ CRITICAL STEP: Sorting the array by date (O(n log n)) allows for fast Binary Search (O(log n)).
    daily_records.sort(key=lambda x: x['date'])

    return new_record # Return the record for the undo stack
```

3. 📝 Interactive Feature: Log New Workout

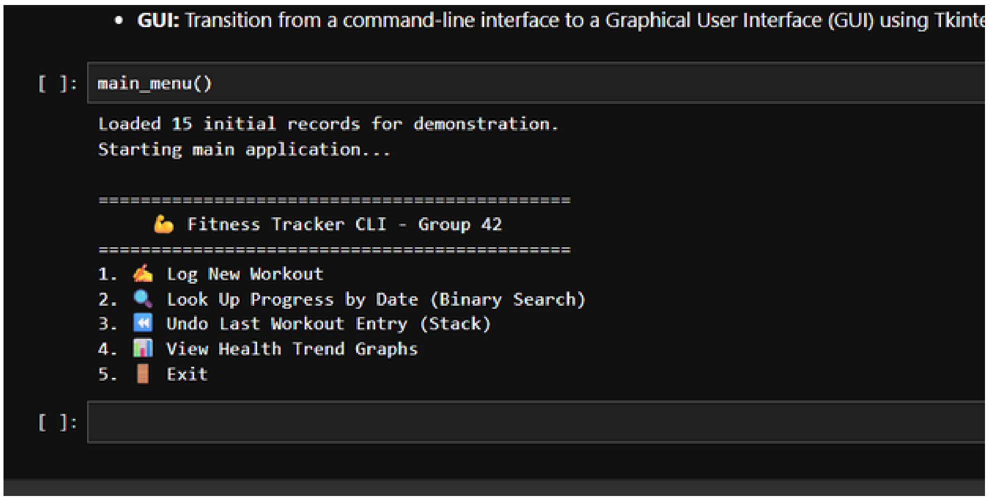
[5]:
```python
# 3. Interactive Feature: Log New Workout

def log_new_workout():
    """
    Guides the user through entering a new daily record using safe input functions.
    """
    print("\n--- ✍️ Log New Workout ---")

    # 1. Get Date
    workout_date = get_date_input()

    # 2. Get Duration (Integer)
    duration = get_valid_input("⏱️ Enter workout duration in minutes: ", data_type=int)
```



4. ⏪ Feature: Stack for Workout Undo

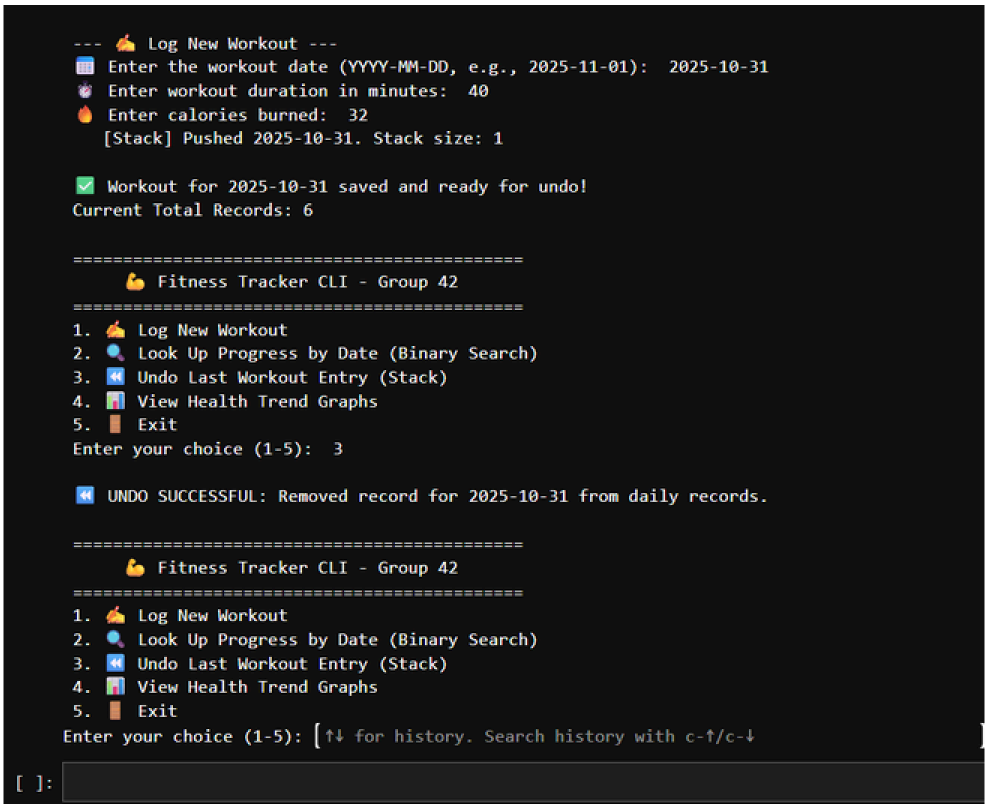
[6]:
```python
# 4. Feature: Workout Undo using STACK

def push_to_undo_stack(record):
    """Pushes a record (or the action details) onto the undo stack."""
    undo_stack.append(record)
    print(f"   [Stack] Pushed {record['date']}. Stack size: {len(undo_stack)}")

def pop_from_undo_stack():
    """Pops the last record from the undo stack and performs the undo action."""
    if not undo_stack:
        print("\n❌ Undo stack is empty. Nothing to undo.")
        return None

    last_record_to_undo = undo_stack.pop()

    # The 'undo' logic: remove the record from the main array
    try:
        daily_records.remove(last_record_to_undo)
        print(f"\n⏪ UNDO SUCCESSFUL: Removed record for {last_record_to_undo['date']} from daily records.")
    except ValueError:
        print(f"\n⚠️ WARNING: Record for {last_record_to_undo['date']} was already removed.")

    return last_record_to_undo

# --- Demonstration Run ---
print("\n--- Running Undo Demonstration ---")
```

```
--- Running Undo Demonstration ---
```

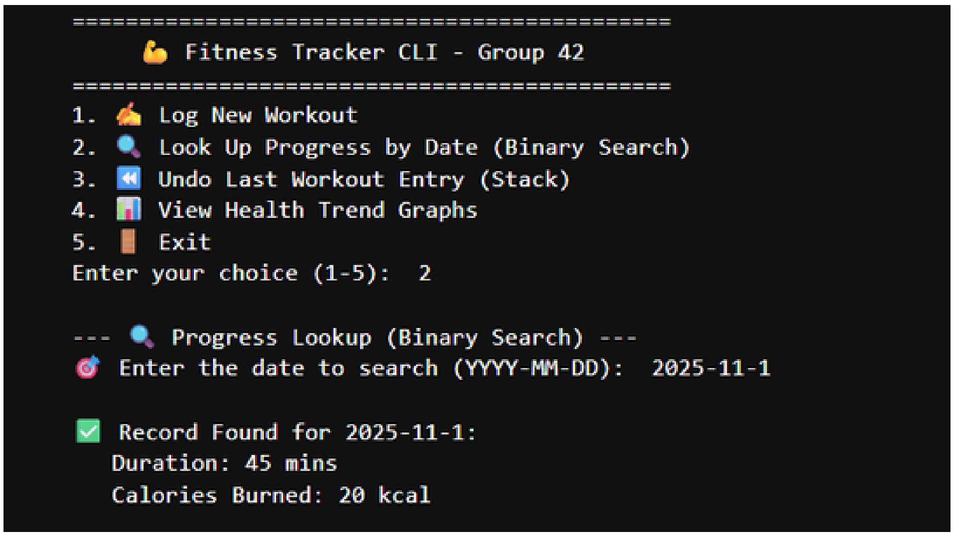


- **GUI:** Transition from a command-line interface to a Graphical User Interface (GUI) using Tkinte

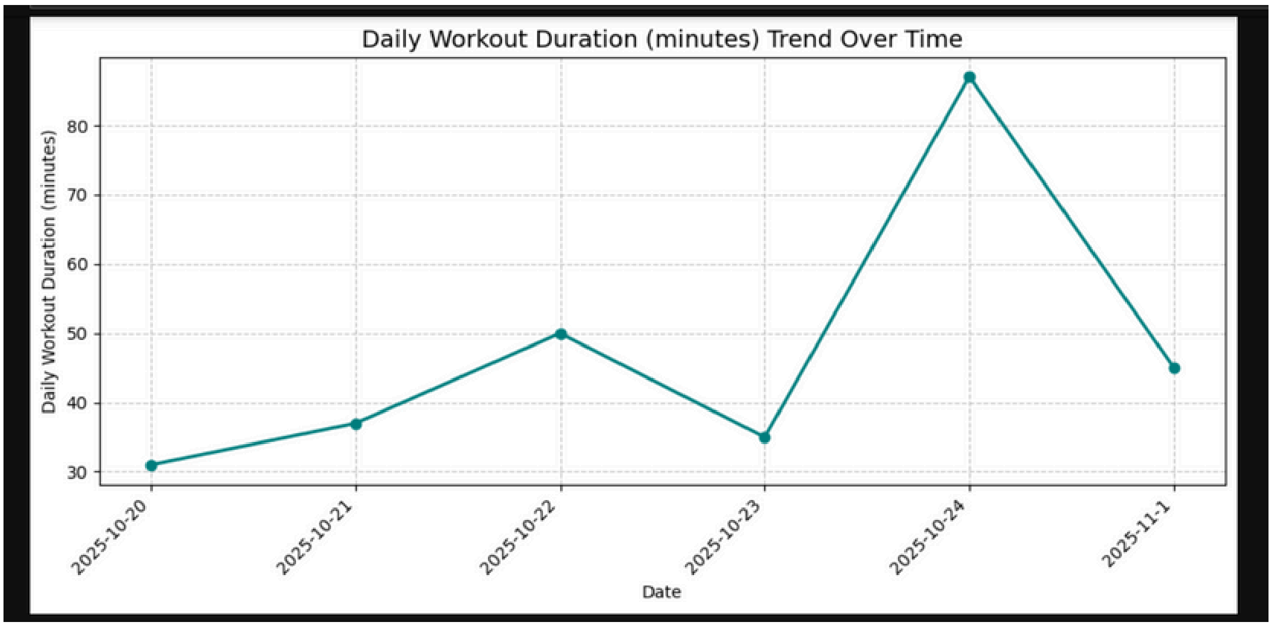
```
[ ]: main_menu()

     Loaded 15 initial records for demonstration.
     Starting main application...

     ==============================================
         💪 Fitness Tracker CLI - Group 42
     ==============================================
     1. ✍️ Log New Workout
     2. 🔍 Look Up Progress by Date (Binary Search)
     3. ⏪ Undo Last Workout Entry (Stack)
     4. 📊 View Health Trend Graphs
     5. 🚪 Exit

[ ]:
```

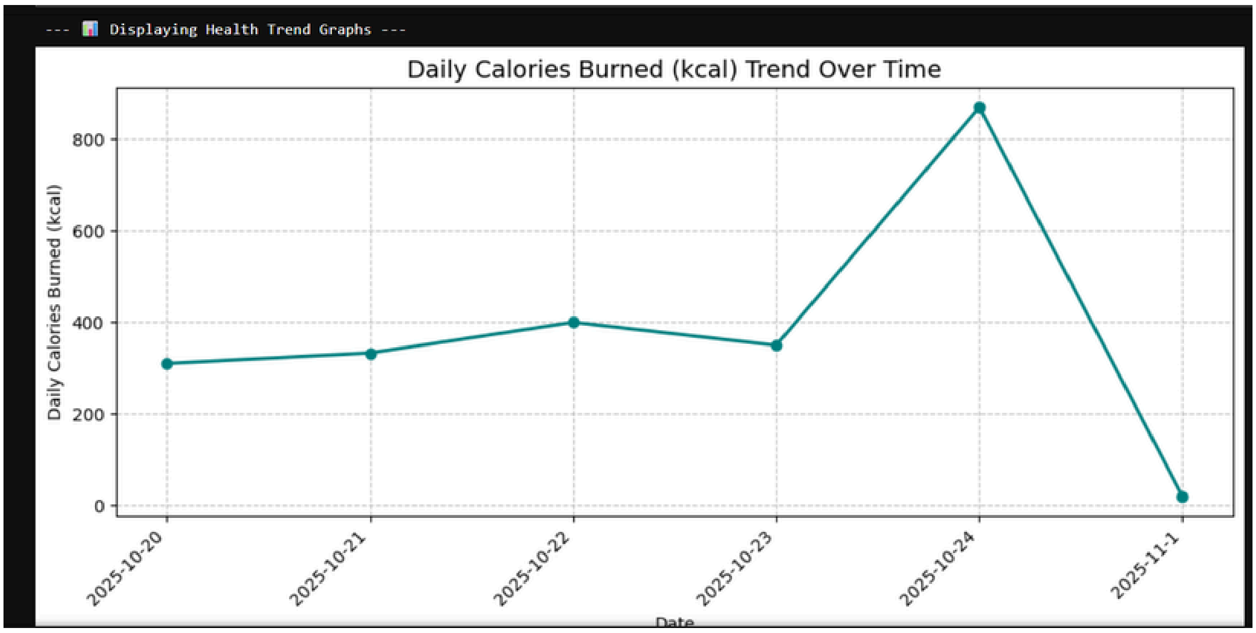


```
     --- ✍️ Log New Workout ---
     📅 Enter the workout date (YYYY-MM-DD, e.g., 2025-11-01):  2025-10-31
     ⏱️ Enter workout duration in minutes:  40
     🔥 Enter calories burned:  32
        [Stack] Pushed 2025-10-31. Stack size: 1

     ✅ Workout for 2025-10-31 saved and ready for undo!
     Current Total Records: 6

     ==============================================
         💪 Fitness Tracker CLI - Group 42
     ==============================================
     1. ✍️ Log New Workout
     2. 🔍 Look Up Progress by Date (Binary Search)
     3. ⏪ Undo Last Workout Entry (Stack)
     4. 📊 View Health Trend Graphs
     5. 🚪 Exit
     Enter your choice (1-5):  3

     ⏪ UNDO SUCCESSFUL: Removed record for 2025-10-31 from daily records.

     ==============================================
         💪 Fitness Tracker CLI - Group 42
     ==============================================
     1. ✍️ Log New Workout
     2. 🔍 Look Up Progress by Date (Binary Search)
     3. ⏪ Undo Last Workout Entry (Stack)
     4. 📊 View Health Trend Graphs
     5. 🚪 Exit
    Enter your choice (1-5): ↑↓ for history. Search history with c-↑/c-↓

[ ]:
```

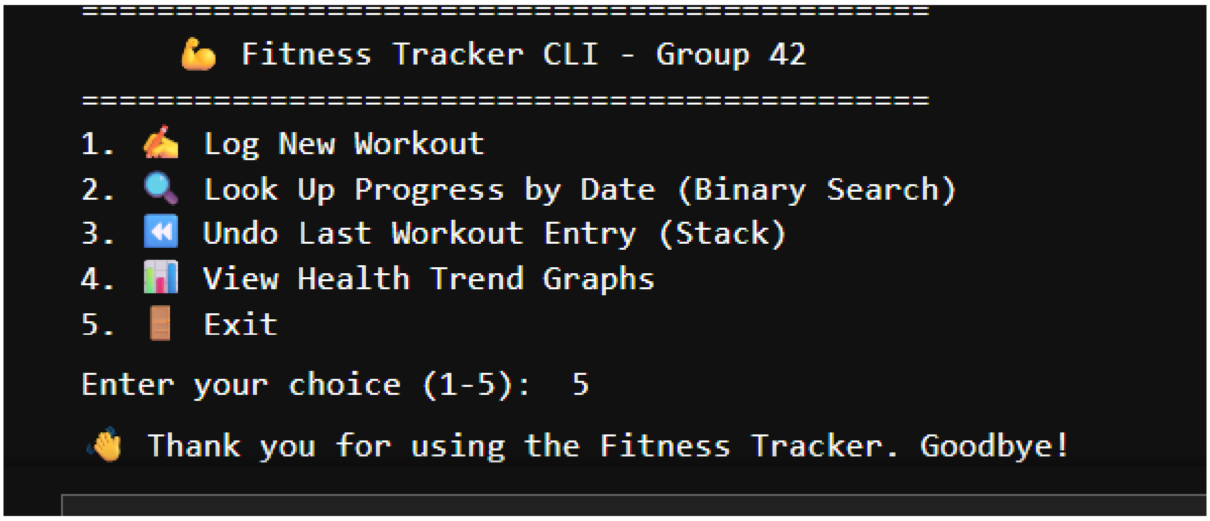


```
==============================================
     💪 Fitness Tracker CLI - Group 42
==============================================
1. ✍️ Log New Workout
2. 🔍 Look Up Progress by Date (Binary Search)
3. ⏪ Undo Last Workout Entry (Stack)
4. 📊 View Health Trend Graphs
5. 🚪 Exit
Enter your choice (1-5):  2

--- 🔍 Progress Lookup (Binary Search) ---
🎯 Enter the date to search (YYYY-MM-DD):  2025-11-1

✅ Record Found for 2025-11-1:
   Duration: 45 mins
   Calories Burned: 20 kcal
```





```
--- 📊 Displaying Health Trend Graphs ---
```



```
============================================
      💪 Fitness Tracker CLI - Group 42
============================================
1. ✍️ Log New Workout
2. 🔍 Look Up Progress by Date (Binary Search)
3. ⏪ Undo Last Workout Entry (Stack)
4. 📊 View Health Trend Graphs
5. 🚪 Exit
Enter your choice (1-5):  5
👋 Thank you for using the Fitness Tracker. Goodbye!
```