

SIMPLE NOTE ORGANIZER

A Terminal-Based Note Management System

Project Report

Submitted by: Harsimran Singh

Roll No: 25BAI10007

Course: Problem Solving and Programming

Department: CSE (AIML)

VIT Bhopal

November 2025

Table of Contents

1. Introduction
2. Problem Statement
3. Functional Requirements
4. Non-functional Requirements
5. System Architecture
6. Design Diagrams
7. Design Decisions & Rationale
8. Implementation Details
9. Screenshots / Results
10. Testing Approach
11. Challenges Faced
12. Learnings & Key Takeaways
13. Future Enhancements
14. References

1. Introduction

In today's digital age, keeping track of information and ideas is crucial for productivity. While there are numerous note-taking applications available, many of them are either too complex, require internet connectivity, or come with unnecessary features that bloat the system. The Simple Note Organizer was developed to address this gap by providing a lightweight, offline, terminal-based solution for personal note management.

This project is a Python-based application that allows users to create, store, search, and manage notes directly from the command line interface. One of the unique features is the ability to attach image file references to notes, making it useful for students, developers, and professionals who need quick access to their thoughts and visual references.

The application stores all data locally in a plain text file format, ensuring complete privacy and eliminating dependency on cloud services or databases. It's designed to be simple enough for beginners to understand while being practical enough for daily use.

1.1 Motivation

The motivation behind this project came from personal frustration with existing note-taking apps that either required an internet connection, had slow startup times, or were overcomplicated for simple note-taking tasks. As someone who spends a lot of time in the terminal, I wanted something that could be accessed quickly without switching contexts or opening heavy GUI applications.

1.2 Scope

The scope of this project includes developing a fully functional command-line note organizer with create, read, search, and delete operations. The application supports attaching image references and provides persistent storage using a custom file format. The project intentionally avoids using external libraries or databases to keep it lightweight and easy to deploy.

2. Problem Statement

Many professionals and students need a quick way to jot down notes, ideas, or information throughout their day. However, existing solutions present several challenges:

- **Complexity:** Most modern note-taking apps come with features like collaboration, cloud sync, and rich text formatting that add unnecessary complexity for personal use.
- **Privacy Concerns:** Cloud-based solutions raise privacy concerns as notes are stored on external servers.
- **Performance:** GUI-based applications can be resource-intensive and slow to launch, especially on older hardware.
- **Internet Dependency:** Many popular solutions require internet connectivity, making them unavailable in offline scenarios.
- **Platform Lock-in:** Proprietary formats make it difficult to migrate data or access notes without specific software.

2.1 Proposed Solution

The Simple Note Organizer addresses these problems by providing:

- A lightweight terminal application that launches instantly
- Complete offline functionality with local storage
- Simple text-based storage format that's human-readable
- Cross-platform compatibility (Windows, macOS, Linux)
- Zero external dependencies beyond standard Python libraries
- Ability to reference image files without copying them

3. Functional Requirements

3.1 Core Features

FR1: Note Creation

The system shall allow users to create new notes with a title and content. Users should also be able to optionally attach an image file path to the note.

FR2: View All Notes

The system shall display all stored notes with their ID, title, content, and attached image path (if any) in a readable format.

FR3: Search Functionality

The system shall provide a search feature that allows users to find notes by entering keywords. The search should look through both note titles and content, and display all matching results.

FR4: Image Viewing

The system shall allow users to open attached images using the system's default image viewer by specifying the note ID.

FR5: Note Deletion

The system shall allow users to delete notes by specifying the note ID. The deletion should be permanent and update the storage file immediately.

FR6: Data Persistence

The system shall automatically save all notes to a local file and load them when the application starts, ensuring data persists across sessions.

3.2 User Interaction Requirements

- The system shall provide a clear menu-driven interface
- The system shall provide friendly, conversational prompts and feedback
- The system shall handle invalid inputs gracefully with helpful error messages
- The system shall confirm successful operations (save, delete, etc.)

4. Non-functional Requirements

4.1 Performance Requirements

- **Startup Time:** The application should launch in under 2 seconds on standard hardware
- **Response Time:** All operations (create, search, delete) should complete within 1 second for datasets up to 1000 notes
- **Search Performance:** Search operations should return results within 500ms for typical use cases

4.2 Usability Requirements

- The interface should be intuitive enough for users with basic command-line knowledge
- Error messages should be clear and guide users toward resolution
- The application should use conversational language rather than technical jargon

4.3 Reliability Requirements

- Data should be saved immediately after every modification to prevent loss
- The application should handle corrupted data files gracefully
- File I/O operations should include basic error handling

4.4 Portability Requirements

- The application should run on Windows, macOS, and Linux without modification
- Only standard Python libraries should be used to ensure compatibility
- The storage format should be platform-independent

4.5 Maintainability Requirements

- Code should be organized into clear, single-purpose functions
- Variable names should be reasonably descriptive
- The storage format should be human-readable for debugging

4.6 Security Requirements

- All data should be stored locally on the user's machine
- No network communication should occur
- File permissions should be respected according to OS standards

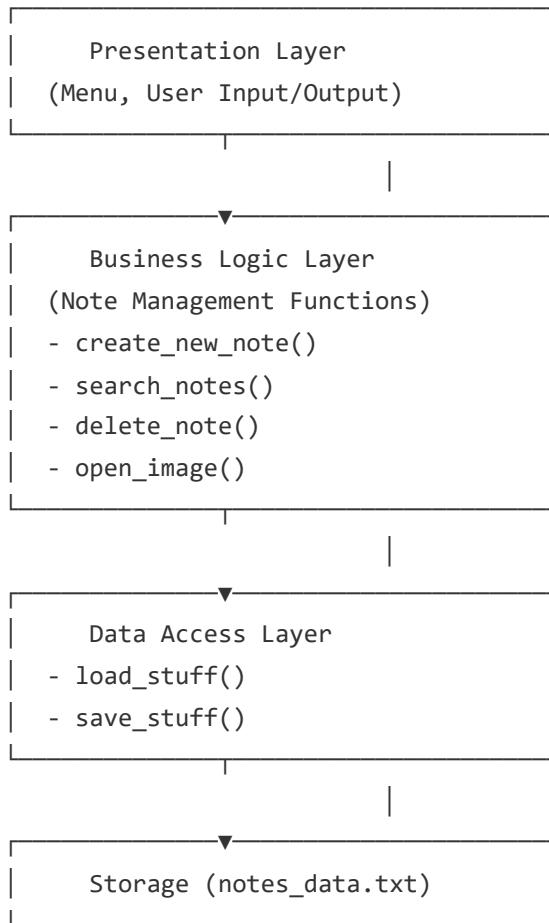
5. System Architecture

5.1 Architecture Overview

The Simple Note Organizer follows a simple layered architecture pattern with three main layers:

1. **Presentation Layer:** Handles user interaction through the terminal interface
2. **Business Logic Layer:** Contains the core functionality for managing notes
3. **Data Access Layer:** Manages reading and writing to the storage file

System Architecture Diagram



5.2 Component Description

5.2.1 Presentation Layer Components

- **show_menu()**: Displays the main menu options to the user
- **main()**: Main loop that processes user input and routes to appropriate functions

5.2.2 Business Logic Components

- **create_new_note()**: Handles note creation with user input validation
- **show_all()**: Formats and displays all notes
- **search_notes()**: Implements keyword search across note titles and content
- **open_image()**: Locates and opens image files using OS-specific commands
- **delete_note()**: Removes notes from the in-memory list and persists changes

5.2.3 Data Access Components

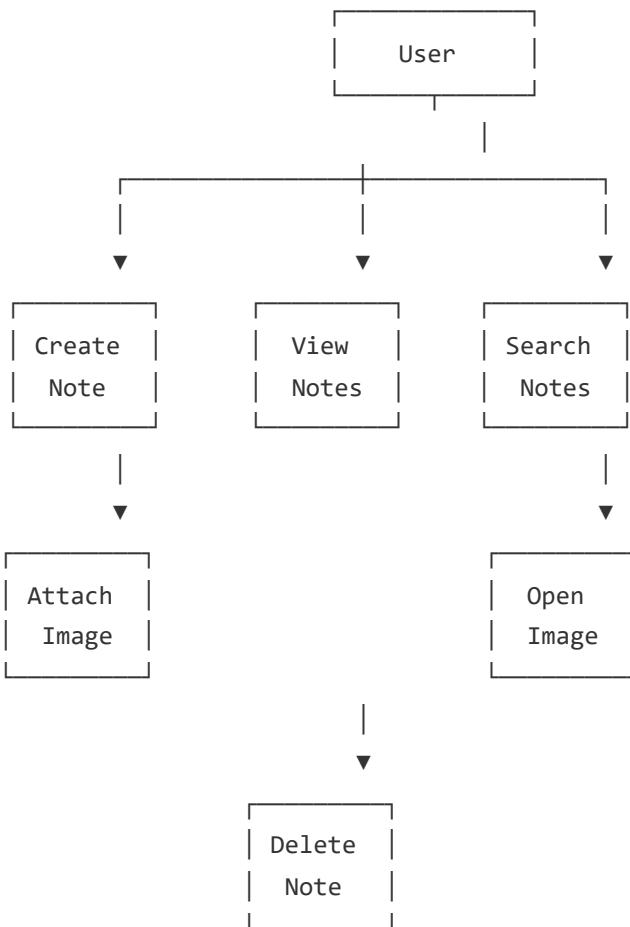
- **load_stuff()**: Reads and parses the storage file at application startup
- **save_stuff()**: Writes the current note list to the storage file

5.3 Data Flow

When the application starts, it loads existing notes from the file into memory. All operations (create, search, view) work with this in-memory list. Whenever a modification occurs (create or delete), the entire list is immediately written back to the file. This approach ensures data consistency while keeping the implementation simple.

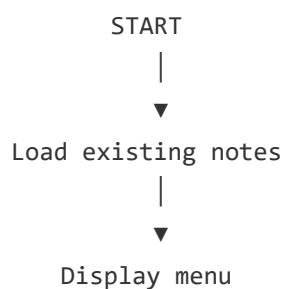
6. Design Diagrams

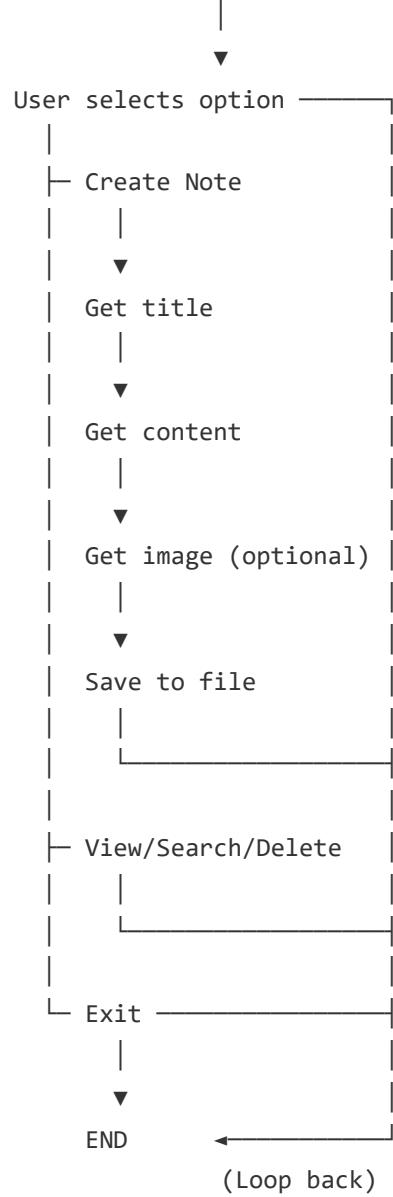
6.1 Use Case Diagram



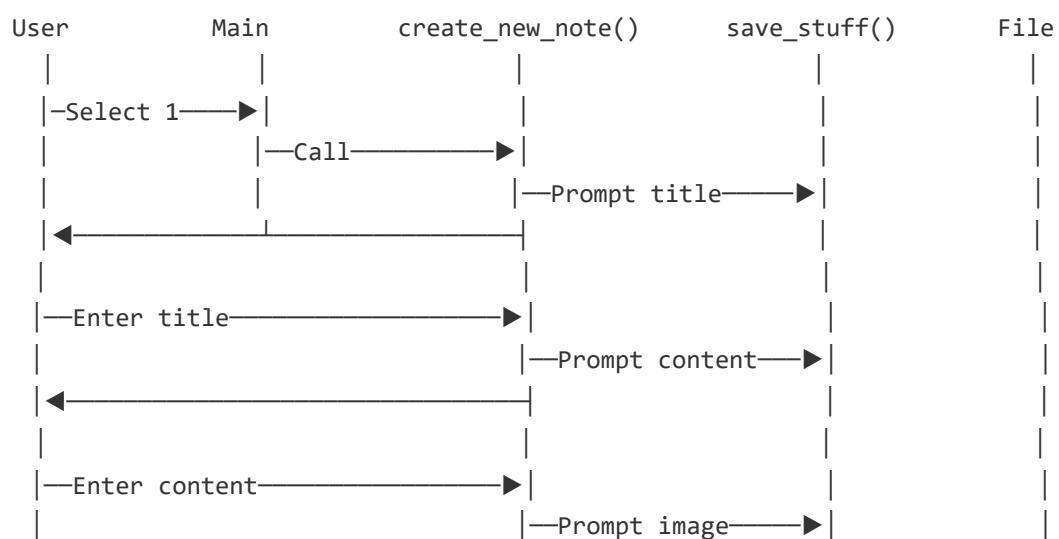
6.2 Workflow Diagram

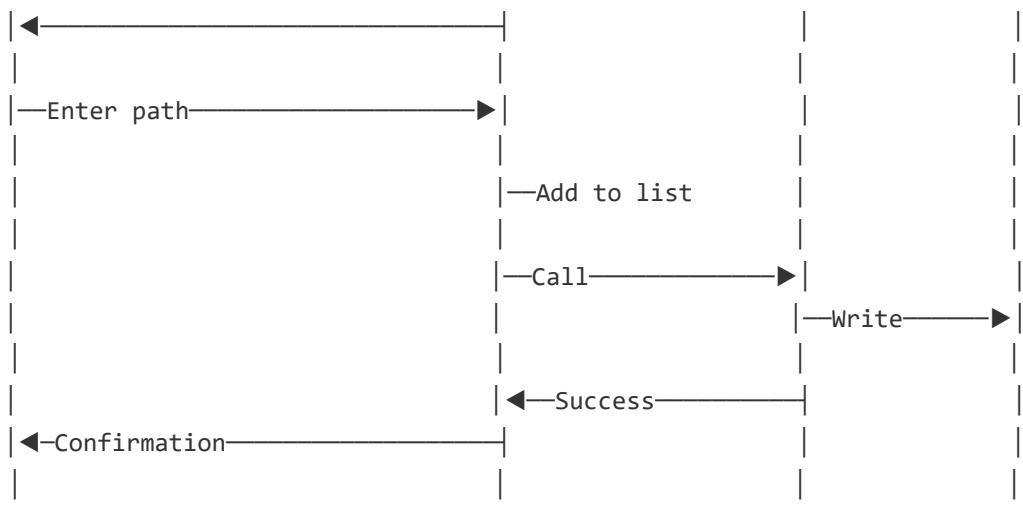
Note Creation Workflow



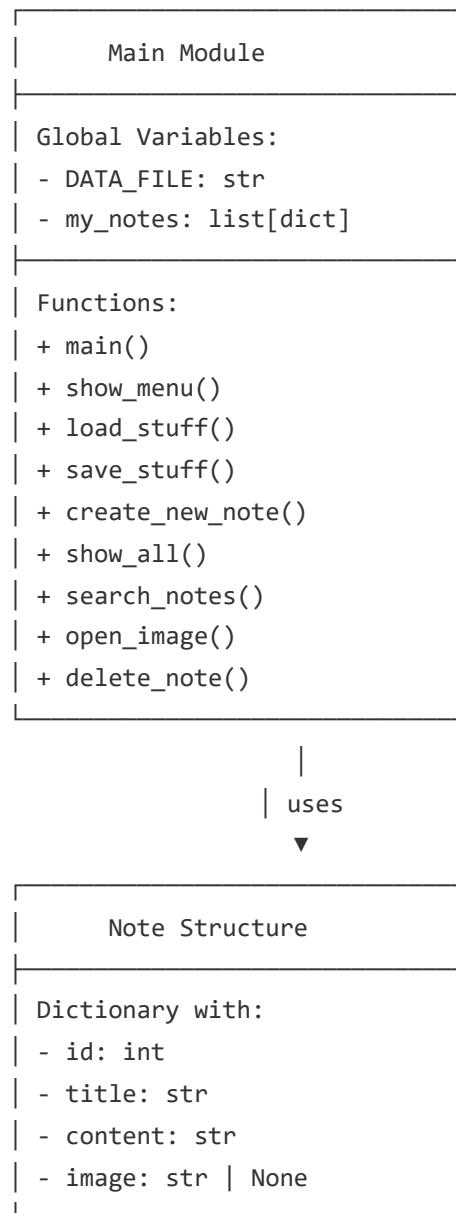


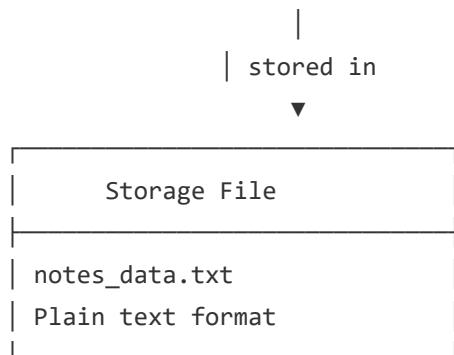
6.3 Sequence Diagram - Creating a Note



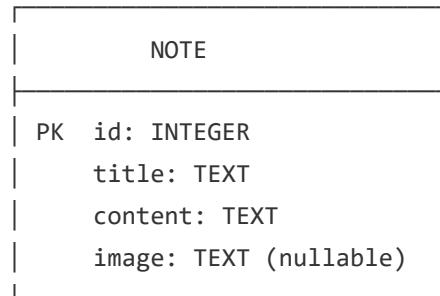


6.4 Class/Component Diagram





6.5 ER Diagram (Data Storage)



Note: This is a flat file storage with no relationships. Each note is independent.

7. Design Decisions & Rationale

7.1 Choice of Python

Decision: Use Python as the programming language.

Rationale: Python was chosen because it's widely available, easy to read and write, and comes with excellent built-in libraries for file handling and OS operations. It also works across different platforms without modification, which was important for this project.

7.2 Text File Storage

Decision: Use a simple text file instead of a database.

Rationale: Initially, I considered using SQLite, but decided against it for several reasons. First, it would add complexity that isn't necessary for the expected data volume. Second, a text file is human-readable, which makes debugging easier. Third, it eliminates any external dependencies. The performance difference is negligible for typical usage (under 1000 notes), and the text file can easily be backed up or version-controlled.

7.3 In-Memory Data Structure

Decision: Load all notes into memory on startup and work with the in-memory list.

Rationale: This approach simplifies the code significantly and provides excellent performance for operations like search and display. The trade-off is memory usage, but even 10,000 notes would only consume a few megabytes. For typical personal use (tens to hundreds of notes), this is completely acceptable.

7.4 Image Reference vs Storage

Decision: Store only file paths to images, not the images themselves.

Rationale: Storing images as binary data would significantly complicate the storage format and increase file size. By storing only paths, we keep the system simple and let the OS handle image viewing with native applications. The downside is that users need to manage image files separately, but this seemed like an acceptable trade-off for simplicity.

7.5 Menu-Driven Interface

Decision: Use a numbered menu system rather than command-line arguments.

Rationale: While command-line arguments (like `note.py add "Title" "Content"`) would be more "Unix-like," a menu-driven approach is more beginner-friendly and doesn't require users to remember syntax. It also makes the application more interactive and conversational.

7.6 Immediate Save Strategy

Decision: Save to disk immediately after every modification.

Rationale: This ensures data integrity even if the program crashes or is terminated unexpectedly. The performance impact is minimal since file I/O is fast for text files of this size, and data safety is more important than the slight performance cost.

7.7 Conversational UI Language

Decision: Use friendly, conversational language in prompts and messages.

Rationale: Technical language can be intimidating for some users. By using phrases like "What should we call this note?" instead of "Enter note title:", the application feels more approachable and less like a stern command-line tool.

7.8 Cross-Platform Image Opening

Decision: Use OS-specific commands to open images rather than a Python library.

Rationale: This approach uses native OS capabilities and respects user preferences for default image viewers. While it makes the code slightly more complex with OS detection, it provides a better user experience than trying to implement image viewing in Python.

8. Implementation Details

8.1 Development Environment

- **Language:** Python 3.x
- **Development OS:** Windows 10
- **Editor:** VS Code
- **Version Control:** Git

8.2 File Structure

```
project/ | └── note_organizer.py # Main application file └── notes_data.txt # Data storage (created on first run) └── README.md # Documentation
```

8.3 Storage Format

Notes are stored in a custom text format that's both human-readable and easy to parse:

```
ID:1 Title:Meeting Notes Content:Discussed project timeline and deliverables  
Image:/path/to/screenshot.png --- ID:2 Title:Shopping List Content:Milk, eggs,  
bread, coffee Image:None ---
```

8.4 Key Implementation Choices

8.4.1 Data Loading

The `load_stuff()` function reads the entire file and parses it line by line. It looks for lines starting with "ID:" to identify the beginning of each note. This simple parsing strategy works well because the format is consistent.

```
def load_stuff(): global my_notes if not os.path.exists(DATA_FILE): return file =  
open(DATA_FILE, 'r') lines = file.readlines() file.close() i = 0 while i <  
len(lines): if lines[i].startswith("ID:"): # Parse note fields... i += 5 else: i  
+= 1
```

8.4.2 Search Implementation

The search is implemented as a simple case-insensitive substring match. While not as sophisticated as fuzzy matching or ranking algorithms, it's fast and works well for the typical use case.

```
def search_notes(): search_term = input("\nWhat are you looking for? ").lower()
results = [] for note in my_notes: title_lower = note['title'].lower()
content_lower = note['content'].lower() if search_term in title_lower or
search_term in content_lower: results.append(note)
```

8.4.3 ID Generation

Note IDs are generated by finding the maximum existing ID and adding 1. This ensures unique IDs even after deletions. There's a slight inefficiency here ($O(n)$ to find max), but it's acceptable for the expected data size.

```
if len(my_notes) == 0: next_id = 1 else: next_id = max([note['id'] for note in
my_notes]) + 1
```

8.4.4 Error Handling

Error handling is kept minimal but practical. File I/O operations are wrapped in try-except blocks, and invalid user input (like non-numeric IDs) is caught with ValueError exceptions.

8.5 Platform-Specific Code

The only platform-specific code is for opening images:

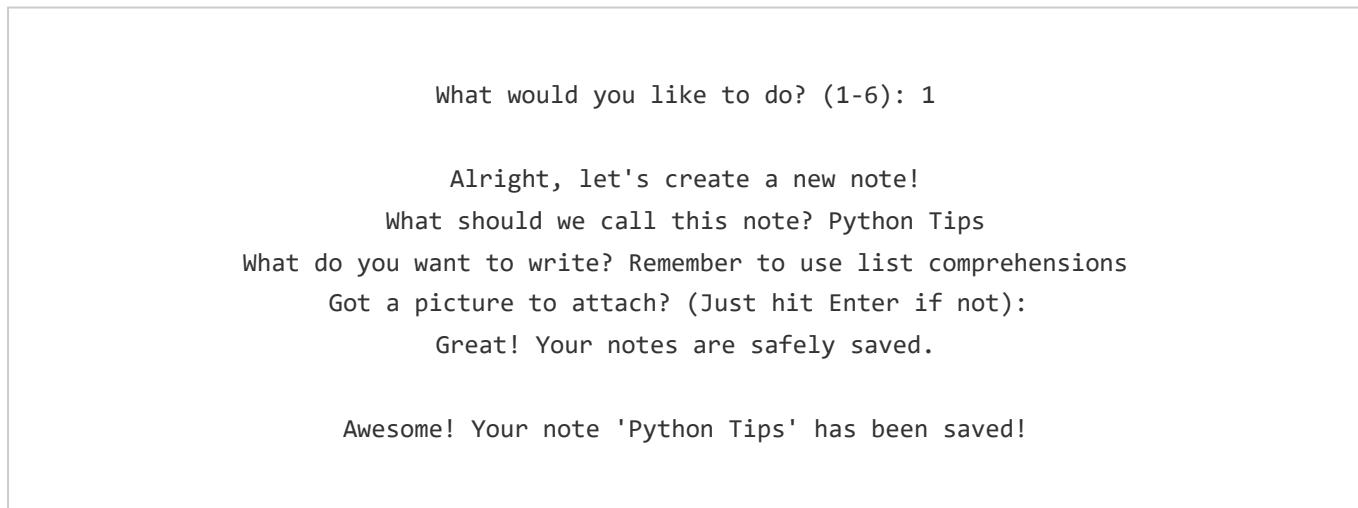
```
if os.name == 'nt': # Windows os.system(f'start "" "{note["image"]}")' elif
os.name == 'posix': # Mac/Linux if os.uname().sysname == 'Darwin':
os.system(f'open "{note["image"]}")' else: os.system(f'xdg-open "'
{note["image"]}")'
```

9. Screenshots / Results

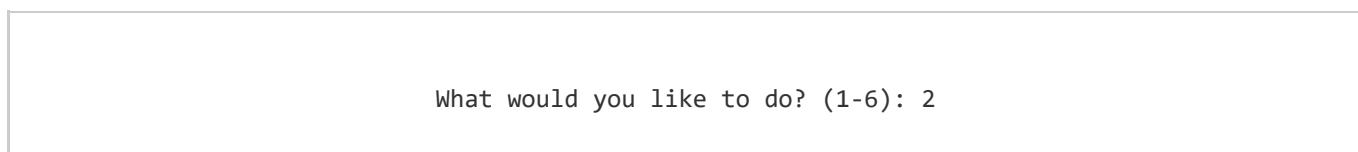
9.1 Main Menu



9.2 Creating a Note



9.3 Viewing All Notes



--- Here are all your notes ---

#1 - Meeting Notes

Discussed project timeline and deliverables for Q4

 /home/user/screenshots/meeting.png

#2 - Shopping List

Milk, eggs, bread, coffee, tea bags

#3 - Python Tips

Remember to use list comprehensions

9.4 Search Results

What would you like to do? (1-6): 3

What are you looking for? python

--- Found 1 note(s) matching 'python' ---

#3 - Python Tips

Remember to use list comprehensions
