

A
MINI-PROJECT REPORT ON
“A Smart Text Editor”

Submitted to
Savitribai Phule Pune University, Pune
In Partial Fulfilment of
S.E. Computer Engineering
Semester IV

Submitted by

Mr. Shubham Avhad
Mr. Sanket Barkade
Mr. Sanket Burke

Under Guidance of
Prof. Sneha Tirth



Department of Computer Engineering
KJEI'S
Trinity College of Engineering and Research, Pune-48
Academic Year 2024-25



TRINITY COLLEGE OF ENGINEERING AND RESEARCH, PUNE

(Accredited by NAAC with A+ Grade Approved by AICTE & Affiliated to SPPU, Pune)
Sr. No.25 & 27, Near Khadi Machine Chowk, Kondhwa Annexe, Pune-48, Maharashtra

Department of Computer Engineering

CERTIFICATE

This is to certify that the **Mini-Project report “A Smart Text Editor”** being submitted by **Mr. Sanket Burke, Mr. Shubham Avhad, Mr. Sanket Burke** a bonafide work carried out by her under the supervision and guidance of **Prof. Sneha Tirth** in fulfilment of the requirement for **SE Computer Engineering** course of Savitribai Phule Pune University, Pune in the academic year 2024-2025.

Prof. Sneha Tirth

Subject Teacher

Dr. Geetika Narang

HOD Computer Engineering TCOER, Pune

Date: 18-04-2025

Place: Pune

ACKNOWLEDGEMENT

The purpose of the acknowledgments page is to show appreciation to those who contributed to conducting this dissertation work /other tasks and duties related to the report writing. Therefore, when writing the acknowledgments page, you should carefully consider everyone who helped during the research process and show appreciation in the order of relevance. In this regard, it is suitable to show appreciation in a brief manner instead of using strong emotional phrases.

In this part of your work, it is normal to use personal pronouns like” I, my, me” while in the rest of the report this articulation is not recommended. Even when acknowledging family members and friends make sure of using the wording of a relatively formal register. The list of the persons you should acknowledge includes the guide (main and second), academic staff in your department, technical staff, reviewers, companies, family, and friends.

You should acknowledge all sources of funding. It’s usually specific naming the person and the type of help you received. For example, an advisor who helped you conceptualize the project, someone who helped with the actual building or procedures used to complete the project, someone who helped with computer knowledge, someone who provided raw materials for the project, etc.

Students Names :

Sanket Burke

Shubham Avhad

Sanket Barkade

INDEX

Sr. No.	Contents	Page No.
1	Abstract	5
2	Introduction	6
3	Problem Statement	7
4	Objectives	8
5	System Design and Theory	9
6	Implementation Code & Output	11
7	Conclusion	14
8	References	15

Abstract

This mini-project revolves around the design and implementation of a Smart Text Editor, a modern tool that combines intuitive functionalities with advanced programming techniques to improve the text editing experience. The primary aim of the project is to enhance user productivity and efficiency by integrating intelligent features such as autocomplete and syntax highlighting.

The autocomplete feature provides real-time word suggestions based on predefined keywords, significantly reducing typing effort and minimizing errors. The syntax highlighting feature visually distinguishes programming components like keywords, strings, and comments, making the editing process seamless and more structured. Together, these features cater to the needs of developers and other users, making the editor more responsive and user-friendly.

Developed using Python and the tkinter library, the project leverages simple yet powerful tools to build an interactive graphical user interface (GUI). The project focuses on creating a scalable and modular application that not only fulfills the immediate requirements of a text editor but also provides a foundation for adding future enhancements. By employing efficient algorithms and real-time event handling, this Smart Text Editor represents a significant leap in text editing tools designed for both practicality and innovation.

Introduction

Text editors are fundamental tools in computing, playing a pivotal role in enabling users to create, edit, and manage textual content effectively. From drafting simple documents to writing complex programming scripts, text editors have become an indispensable part of both personal and professional tasks. However, traditional text editors often lack advanced features that can streamline and optimize the editing experience, especially for users engaged in programming or working with structured text.

This project introduces the concept of a "Smart Text Editor," a next-generation tool designed to enhance the traditional editing experience by integrating intelligent features. Unlike conventional editors, this Smart Text Editor incorporates advanced functionalities such as keyword autocomplete and syntax highlighting. These features are aimed at improving user productivity, reducing errors, and fostering an intuitive editing environment.

The keyword autocomplete feature predicts and suggests commonly used words or phrases based on user input, making the typing process faster and more efficient. It is particularly beneficial for developers who frequently use repetitive keywords or constructs in their code. Meanwhile, the syntax highlighting feature enhances code readability by visually differentiating keywords, strings, comments, and other syntactic elements through color coding, allowing users to quickly understand and debug their code.

Built using Python and the tkinter library, the Smart Text Editor leverages a simple yet powerful graphical user interface to deliver a seamless and interactive user experience. The project also integrates robust backend logic, including regular expressions (regex) for pattern matching and real-time event handling, to ensure smooth functionality.

This mini-project not only showcases the integration of modern programming techniques but also emphasizes the potential of combining user-friendly design with intelligent algorithms. By addressing the needs of both casual users and developers, the Smart Text Editor aims to set a new standard in text editing applications.

Problem Statement

Problem statement :

Design a mini project to implement a Smart text editor

Problem Description :

In the modern era, text editing has become an integral part of daily life, encompassing activities such as document creation, coding, and note-taking. However, traditional text editors often fall short when it comes to addressing common user challenges, such as the inefficiencies caused by repetitive typing and the absence of advanced editing tools. These limitations can lead to wasted time, errors, and a cumbersome user experience, particularly for developers and professionals who rely on text editors for complex tasks.

The need for an innovative solution that streamlines and enhances the editing process has driven the development of this project. The Smart Text Editor is designed to tackle these challenges by introducing intelligent features like autocomplete, which reduces repetitive typing, and syntax highlighting, which enhances readability and organization, especially for programming text. By addressing these pain points, the Smart Text Editor aims to provide an intuitive and efficient platform that improves the overall user experience in text editing.

This project is a response to the growing demand for smarter tools that not only simplify editing tasks but also adapt to the needs of modern users. By leveraging advanced programming techniques and real-time responsiveness, the Smart Text Editor fills the gap left by traditional editors, offering a tailored solution for both everyday and specialized text editing requirements.

Objectives

- To design and implement a Smart Text Editor with enhanced functionality for efficient and user-friendly text editing.
- To develop an autocomplete feature that predicts and suggests frequently used keywords, improving typing speed and reducing repetitive effort.
- To integrate syntax highlighting to visually differentiate programming elements, enhancing readability and assisting in error identification.
- To build a responsive and intuitive graphical user interface (GUI) using Python's tkinter library for seamless interaction.
- To ensure robust file management operations, such as creating, opening, saving, and editing text documents.
- To explore and utilize Python libraries and tools, including regex, for advanced text analysis and processing.

Theory

The concept of a Smart Text Editor revolves around enhancing traditional text editing functionalities with intelligent features that streamline the user experience. These advanced capabilities include autocomplete suggestions and syntax highlighting, which improve efficiency and accuracy during text editing, especially for programming tasks.

1. Autocomplete

Autocomplete is designed to predict and display word suggestions as the user types. This feature is particularly useful for repetitive coding tasks, where commonly used keywords or functions can be quickly inserted. The mechanism involves:

- Maintaining a predefined list of keywords (e.g. **if**, **else**, **def**, **class**).
- Monitoring user input in real-time to identify partially typed words.
- Displaying a dropdown or suggestion box with matching options.
- Allowing the user to select a suggestion, which automatically completes the word in the text editor.

The implementation relies on string matching and the efficient handling of keyboard events to provide real-time responsiveness. By reducing typing effort and preventing errors, autocomplete significantly enhances productivity.

2. Syntax Highlighting

Syntax highlighting visually differentiates elements of the text based on their function or role within a programming language. For instance:

- **Keywords** (e.g. **if**, **class**, **return**) are displayed in distinct colors to emphasize their importance.
- **Strings** enclosed in quotes are often colored in green to set them apart from other code components.
- **Comments** are shown in a muted or gray color, indicating they are not executed as part of the code.

The feature uses **regular expressions (regex)** to detect patterns within the text, such as keywords, strings, or comments, and applies specific styles to each. Tags are dynamically added and removed as the user types to ensure accurate syntax representation.

3. Graphical User Interface (GUI)

The Smart Text Editor leverages the **tkinter library** to create an intuitive and interactive graphical user interface. This includes:

- A **menu bar** for accessing file operations like New, Open, Save, and Exit.
- Editing options such as **Cut**, **Copy**, and **Paste**.
- A **scrollable text area** to handle large documents seamlessly.

4. Integration of Features

The editor harmoniously combines the above functionalities by:

- Using an event-driven approach to trigger autocomplete and **syntax highlighting** while typing.
- Ensuring the features operate without compromising the editor's performance or responsiveness.

5. Technologies Used

The Smart Text Editor is implemented using:

- **Python** for its simplicity and rich library support.
- **tkinter** for building the graphical interface and managing user interactions.
- **Regex** for text pattern detection in autocomplete and syntax highlighting.

Overall, the theory highlights the practical integration of programming logic, user interface design, and real-time event handling to create an intuitive Smart Text Editor that meets modern-day requirements for productivity tools.

Implementation Code & Output

Code Overview :

The implementation of the Smart Text Editor is divided into several key modules, each handling a specific aspect of the application. Below is a detailed breakdown of the code structure:

1. **editor.py** :

- This module forms the core of the text editor. It includes essential functionalities such as creating, opening, saving, and editing text files.
- Features like **cut_text**, **copy_text**, and **paste_text** simplify text manipulation, while file handling functions (**open_file**, **save_file**, and **save_file_as**) ensure seamless file management.
- The use of tkinter's text widget makes the text area interactive and supports advanced operations like undo functionality and word wrapping.

2. **smart_features.py** :

- This module introduces the "smart" aspect of the text editor. It incorporates:
 - **Autocomplete**: A real-time feature that predicts and suggests words as the user types, improving typing efficiency and reducing errors
 - **Syntax Highlighting**: A functionality that color-codes elements like keywords, strings, and comments for better readability and structure comprehension.
- Leveraging tkinter and regex, these features operate in harmony with the editor, enhancing the user experience.

3. **main.py** :

- Acts as the entry point for the application, integrating the core editor (**editor.py**) and smart features (**smart_features.py**) into a cohesive graphical user interface.
- It defines the menu bar with File and Edit options, allowing users to perform various operations like creating new files, opening existing ones, and accessing editing tools.
- The tkinter framework is used extensively to build an intuitive and interactive interface.

4. **test_editor.py** :

- This script is designed for testing purposes. It initializes the Smart Text Editor application and verifies its features and functionalities.

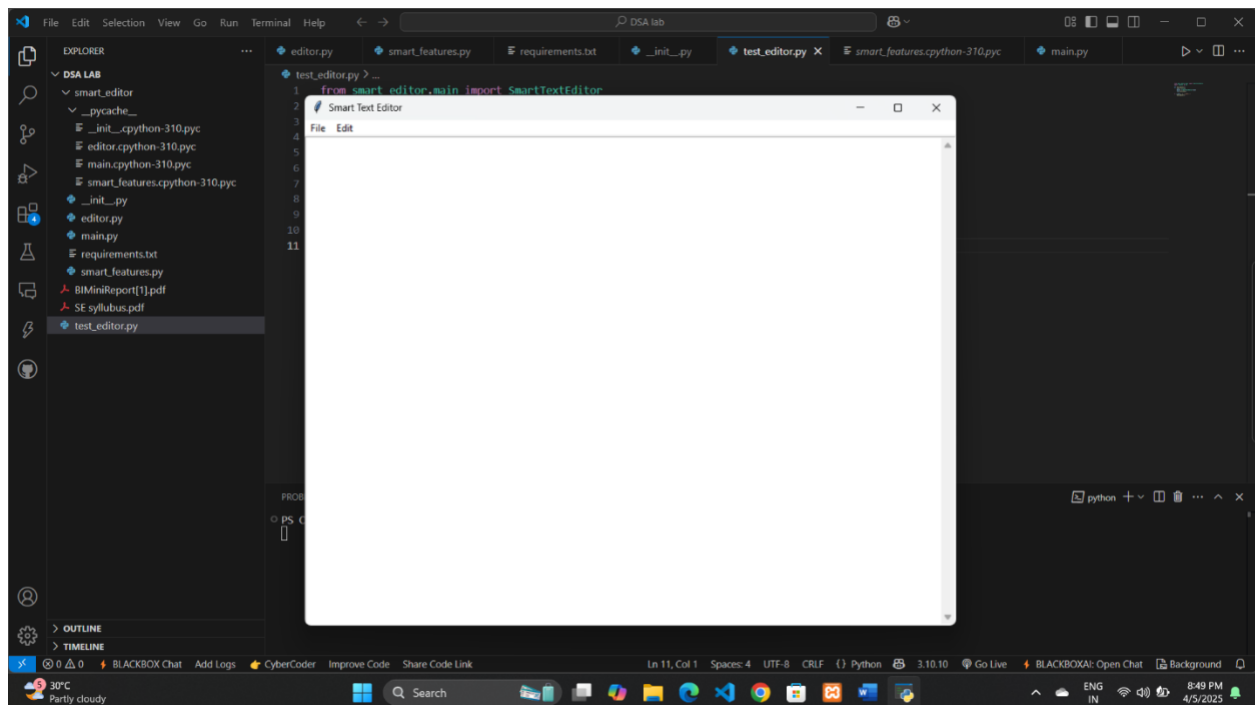
- By running this module, developers can ensure that all components of the editor work seamlessly together.

5. requirements.txt :

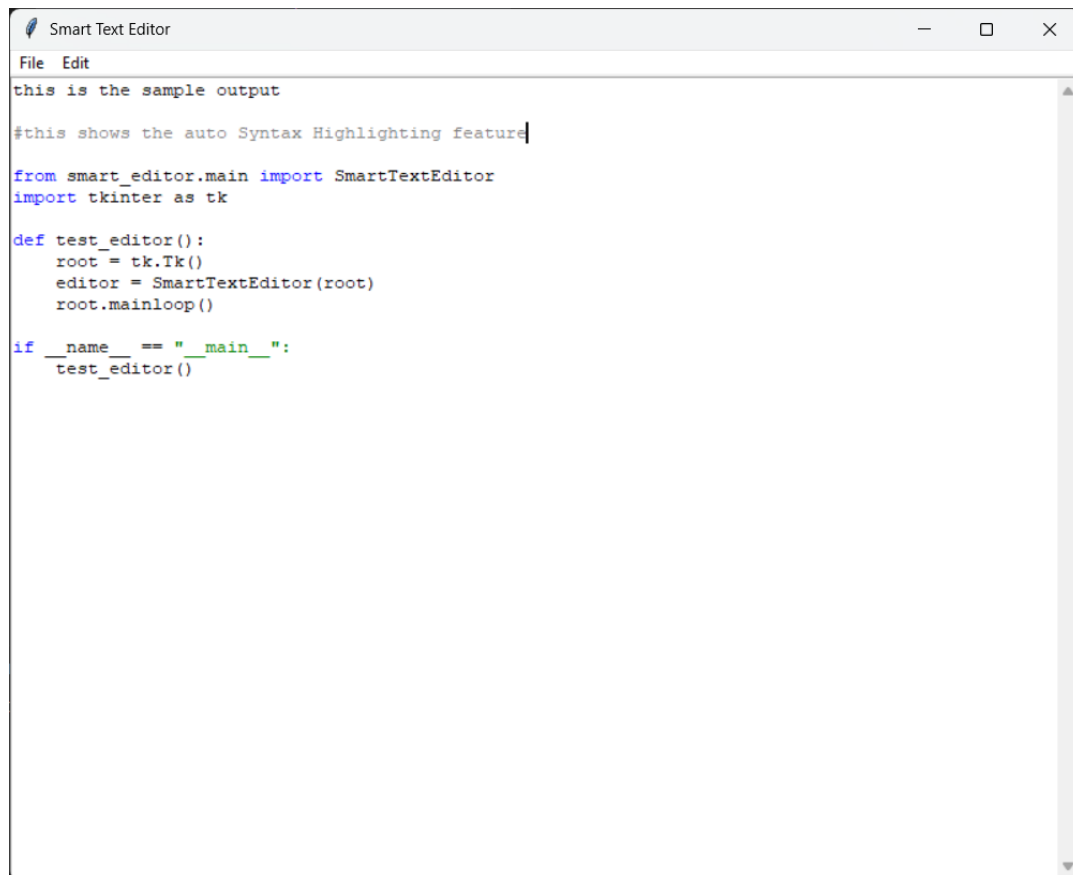
- Contains the dependencies for the project, primarily listing **tkinter** as a requirement to ensure the graphical interface functions properly.

OUTPUT :

1. Main Interface :



2. Syntax Highlighting:



The screenshot shows a window titled "Smart Text Editor" with a menu bar containing "File" and "Edit". The text area contains the following Python code with syntax highlighting: comments are in grey, strings in green, and keywords in blue. The code is as follows:

```
this is the sample output

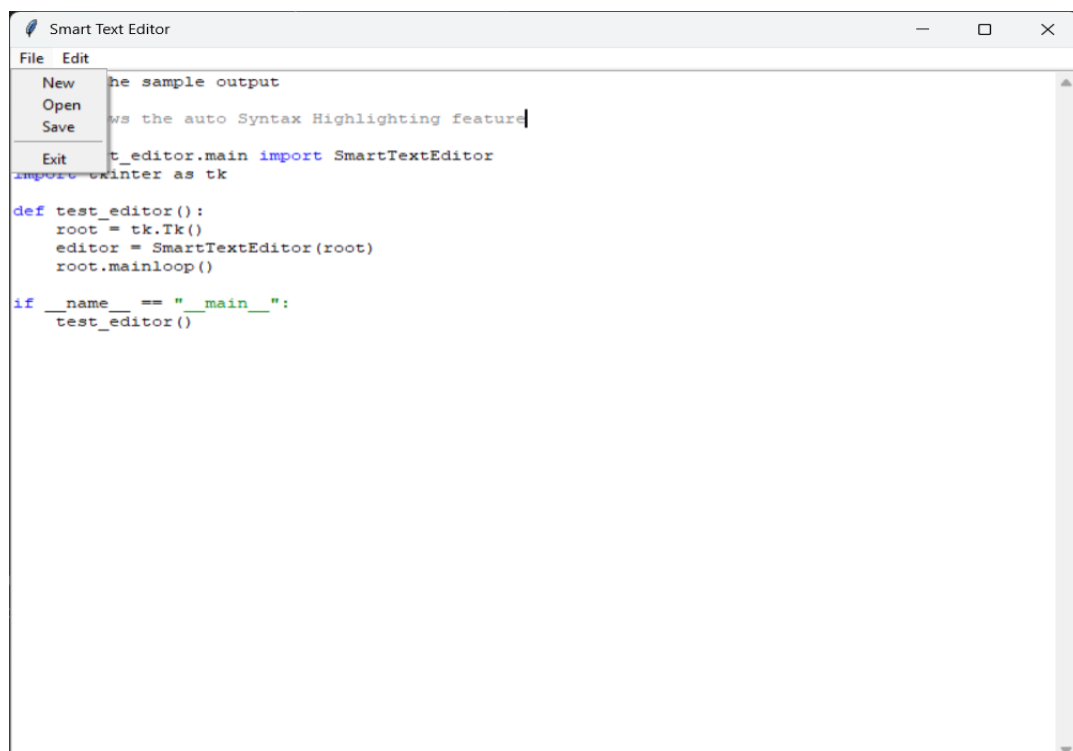
#this shows the auto Syntax Highlighting feature

from smart_editor.main import SmartTextEditor
import tkinter as tk

def test_editor():
    root = tk.Tk()
    editor = SmartTextEditor(root)
    root.mainloop()

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

3. File Operations



The screenshot shows the same "Smart Text Editor" window, but with the "File" menu open. The menu options are "New", "Open", "Save", and "Exit". The text area contains the same Python code as in the previous screenshot, with syntax highlighting.

```
this is the sample output

#this shows the auto Syntax Highlighting feature

from smart_editor.main import SmartTextEditor
import tkinter as tk

def test_editor():
    root = tk.Tk()
    editor = SmartTextEditor(root)
    root.mainloop()

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

Conclusion

The Smart Text Editor represents a significant leap forward in text editing tools, blending user-focused design with cutting-edge programming techniques. By incorporating features such as keyword autocomplete and syntax highlighting, the editor addresses the limitations of traditional text editing software and provides an intuitive and efficient platform for users, especially developers and writers working with structured text.

The successful implementation of the autocomplete feature has demonstrated its ability to save time and effort by predicting commonly used keywords and phrases, reducing typing redundancies. Syntax highlighting has further enhanced readability and organization, allowing users to identify crucial code elements at a glance and facilitating faster debugging.

This project exemplifies the integration of advanced concepts such as real-time event handling, regex-based pattern recognition, and modular programming into a practical application. The use of Python and tkinter not only ensures simplicity and reliability but also opens up opportunities for scalability and future enhancements.

While the current version achieves its primary objectives, there is ample scope for expanding its capabilities. Future improvements could include enlarging the autocomplete library to cover more programming languages and contexts, adding themes for customization, and integrating machine learning models to dynamically adapt suggestions based on user preferences and editing history. Features such as collaborative editing, cloud integration, and advanced error checking could also make the Smart Text Editor even more versatile and impactful.

In conclusion, this project successfully demonstrates the potential of blending intelligent features with user-friendly design to create an application that significantly enhances productivity and user experience. It serves as a foundation for future developments in the field of text editing tools and showcases the benefits of leveraging programming expertise to solve real-world challenges.

References

1. **Blackbox AI :**

Blackbox AI provides tools for streamlining coding workflows, debugging, and writing effective programming scripts. It was referenced for implementing and optimizing features in the Smart Text Editor.

2. **Copilot AI :**

Microsoft Copilot AI was utilized as a brainstorming and assistance tool for structuring the documentation, refining concepts, and enhancing the overall presentation of the project.