

**Project Title :** “ To-Do List App Using Python”

Group Members :

Muneeb Hamza 12

Ali Zain 05

Uzair Ijaz

Nadeem

Yasir Abbas

Course Title : Data Structure

Instructor Name : Sir. Faisal Hafeez

( IT Incharge )

Submission Date : 09-01-2025

**Acknowledgement :**

We would like to express our deepest gratitude to everyone who contributed to the successful completion of this team project, "To-Do List Application Using Tkinter."

First and foremost, we extend our heartfelt thanks to Our Teachers for their invaluable guidance, constructive feedback, and encouragement throughout the project. Their expertise played a crucial role in shaping the functionality and design of our application.

We also acknowledge the collaborative efforts of our team members, whose dedication, creativity, and teamwork were instrumental in bringing this project to fruition. Each member's contributions, whether in coding, debugging, or design, were vital to its success.

We are also grateful to University oF Layyah for providing the platform necessary for this project. The environment fostered learning, innovation, and seamless collaboration.

This project is a testament to the power of teamwork and shared commitment. We appreciate everyone who contributed directly or indirectly to its success.

**Thank you!**

Summary :

This project, titled "To-Do List Application Using Tkinter," is a user-friendly desktop application designed to help users efficiently manage their daily tasks. Built using Python and the Tkinter library, the application provides a simple yet powerful interface for task management.

The application features a secure login system to ensure user privacy, with access controlled by a password. Once logged in, users can perform the following actions:

* **Add Tasks**: Users can create and save new tasks, which are stored locally for future reference.
* **Update Tasks**: Existing tasks can be edited and updated directly within the application.
* **Mark Important Tasks**: Users can prioritize tasks by marking them as important, visually distinguishing them in the task list.
* **Delete Tasks:** Completed or unnecessary tasks can be removed to keep the list organized.

The application's backend leverages file handling to persist task data, ensuring that all tasks remain accessible even after the application is closed. A responsive graphical user interface **(GUI)** enhances usability, featuring labeled buttons, scrollable task views, and intuitive controls for seamless interaction.

This project not only demonstrates the team's ability to design a functional application but also highlights their proficiency in Python programming, GUI development, and teamwork. It serves as a practical solution for task organization and as a learning milestone for future software development projects.

**Introduction:**

In the modern age of technology, time management and organization are critical for productivity and efficiency. The *"To-Do List Application Using Tkinter"* is a Python-based desktop application designed to simplify task management. This project demonstrates the practical use of Python's Tkinter library to create a graphical user interface (GUI) that offers an intuitive, visually appealing, and functional experience.

The primary objective of this project is to provide a streamlined solution for users to add, update, prioritize, and delete tasks, helping them organize their day effectively. The application includes a secure login feature, ensuring that only authorized users

can access the task list. Tasks are stored persistently using file handling, allowing users to save and retrieve their data seamlessly.

The user-friendly design incorporates various features:

* Task Management: Users can create, edit, and remove tasks as needed.
* Priority Marking: Important tasks can be highlighted for better visibility.
* Responsive Interface: The application employs scrollable views, labeled buttons, and a modern layout for ease of use.
* Security: A password-protected login ensures data confidentiality.

This project not only highlights the application of Tkinter for GUI development but also serves as a valuable learning experience in programming concepts such as event-driven programming, file handling, and user interface design. By addressing real-world challenges in task management, this project illustrates the potential of software solutions to enhance everyday productivity.

**Objective :**

The primary objective of this project is to develop a user-friendly and efficient To-Do List Application using Python's **Tkinter** library. The application is designed to help users organize and manage their daily tasks effectively through a graphical user interface (GUI). The key goals of the project are:

1. **Task Management:**

Provide functionality to add, update, delete, and mark tasks as important, ensuring comprehensive task handling.

1. **User Accessibility:**

Develop a secure login system to restrict unauthorized access and protect user data.

1. **Persistence of Data:**

Implement file handling to save tasks locally, allowing users to access their to-do lists even after restarting the application.

1. **Interactive and Intuitive Interface:**

Create a visually appealing GUI with features such as scrollable task lists, labeled buttons, and intuitive design elements for easy navigation.

1. **Prioritization Features:**

Enable users to mark tasks as important, helping them identify high-priority items at a glance.

1. **Practical Application of Programming Concepts:**

Demonstrate the use of Python's Tkinter library for GUI development and employ concepts such as event-driven programming and file management in a real-world scenario.

The project aims to provide a robust and scalable solution for daily task management while highlighting the versatility and practicality of Python in software development.

**Tools and Technology :**

The development of the To-Do List Application was facilitated using the following tools and technologies:

1. **Programming Language:**

Python:

The core language used for the development of the application, chosen for its

simplicity, readability, and extensive library support.

1. **GUI Library:**

Tkinter:

A built-in Python library used to design and develop the graphical user

interface (GUI) of the application. Tkinter provides tools for creating windows,

labels, buttons, text fields, and other UI components.

1. **File Handling:**

Text File (.txt):

Used for storing tasks persistently. Tasks are saved in a file (tasklist.txt),

ensuring data remains accessible even after the application is closed.

1. **Python Libraries:**

simpledialog:

Provides dialog boxes for user input, such as updating tasks.

Messagebox:

Used for displaying messages to users, including error alerts and

confirmations.

1. **Development Environment:**

IDLE/PyCharm/VS Code:

Any of these IDEs or text editors can be used for writing, debugging,

and running the Python code efficiently.

1. **Graphics and Icons**

PhotoImage:

Used to load and display images for icons and UI design elements,

enhancing the visual appeal of the application.

1. **Concepts and Techniques:**

Event-driven Programming:

The application responds to user actions (e.g., button

clicks) to perform specific tasks.

File I/O Operations:

Reading from and writing to files for task management.

Global Variables:

Used to maintain the task list throughout the application's lifecycle.

Error Handling:

Ensures the application remains robust and handles missing files or

incorrect user input gracefully.

1. **Platform:**

Cross-platform Compatibility:

The application is built using Python and Tkinter,

making it compatible with multiple platforms like

Windows, macOS, and Linux.

1. **Design Approach:**

Object-oriented Programming Concepts:

Though this project uses a functional approach, it

incorporates modular programming to maintain clean

and structured code.

These tools and technologies collectively contribute to the seamless functioning and user-friendly interface of the To-Do List Application.

Methodology :

The To-Do List Application was developed using a systematic and iterative approach to ensure robust functionality, ease of use, and maintainability. The methodology employed includes the following steps:

**Requirement Analysis:**

* The project was initiated by identifying the key functionalities required in a task management application.
* Features such as adding, updating, deleting, marking tasks as important, and saving tasks persistently were outlined.
* A user-friendly graphical interface was prioritized for simplicity and accessibility.

**Design and Planning:**

Interface Design:

The application's layout and design were planned to provide a clean

and intuitive interface using the Tkinter library. Components such as

input fields, buttons, listboxes, and icons were included.

Data Handling:

A text file (tasklist.txt) was chosen for task storage to ensure simplicity

and compatibility without requiring a database.

Development:

The project was divided into functional modules for better

organization and code reuse.

Login Functionality:

A secure login window was created to prevent unauthorized

access. The password validation was implemented using the

Entry widget for user input and messagebox for error handling.

Task Management:

Features for adding, updating, deleting, and marking tasks as

important were implemented. Functions were developed to

interact with the tasklist.txt file for reading and writing tasks

persistently.

Graphical User Interface (GUI):

Tkinter components such as Frame, Text, Listbox, and

Button were utilized to create an interactive GUI.

File Handling:

File I/O operations were employed to ensure tasks are

stored and retrieved from the text file seamlessly.

**Implementation Details :**

1. The **Event-driven programming model** was adopted to handle user actions, such as button clicks, listbox selections, and keyboard inputs.
2. Global variables and functions were used to maintain task data across the application.
3. Icons and images were integrated using PhotoImage to enhance the visual appeal.

**Testing and Debugging:**

Functionality Testing:

Each module (login, adding tasks,

deleting tasks, etc.) was tested

individually to ensure correctness.

Integration Testing:

The interactions between modules

were tested to verify smooth

operation.

Error Handling:

Exceptions, such as file absence or

incorrect user inputs, were handled

gracefully to prevent application

crashes.

**Iteration and Refinement:**

1. Based on testing feedback, the application was iteratively refined to improve usability and fix identified issues.
2. Enhancements like task prioritization (important tasks) and a cleaner layout were added.

**Deployment:**

1. The final application was made platform-independent, ensuring compatibility with Windows, macOS, and Linux.
2. The use of a lightweight storage mechanism (text file) ensured easy deployment without additional dependencies.

This structured methodology ensured the successful development of a functional and user-friendly To-Do List Application.

**Implementation Process :**

The implementation process of the To-Do List Application was carried out systematically to ensure the development of a functional, user-friendly, and efficient program. Below is the step-by-step implementation process:

**Environment Setup:**

The Python programming language was chosen for its simplicity and the availability of the Tkinter library for GUI development.

The Tkinter library was imported, along with essential modules such as simpledialog and messagebox for user interaction and input handling.

An appropriate directory structure was set up to organize code, resources, and icons/images used in the application.

**Login Window Development:**

- A login window was designed as the entry point to the application.

- A password input field using the Entry widget with masking enabled (show="\*").

- Password validation logic to grant or deny access.

- Error handling using the messagebox module to display appropriate messages.

- Once the password was validated, the login window transitioned to the main application window.

**Main Application Window :**

Layout Design:

The main window was created with a title, fixed dimensions, and a custom icon. The layout included sections for task input, task display (via a listbox), and buttons for task operations.

Task Input Section:

1. A Text widget was used for multi-line task input.
2. A button labeled "ADD" was implemented to trigger task addition.

**Task Management Functionalities:**

The application supported core task management operations. Each feature was implemented as a separate function for modularity.

**Add Task:**

* Tasks entered in the input field were saved to a tasklist.txt file and displayed in the listbox.
* The addTask function handled file writing and task display updates.

**Delete Task:**

* Tasks selected in the listbox could be deleted using the "DELETE" button.
* The deleteTask function removed the task from the list and updated the file.

**Update Task:**

* Users could select a task and update its content using the "UPDATE" button.
* The updateTask function used simpledialog for input and updated the list and file.

**Mark Task as Important:**

* Tasks could be marked as important with a star symbol ("⭐") using the "IMPORTANT" button.
* The importantTask function toggled the star symbol for selected tasks.

**File Management:**

* The tasklist.txt file was used for persistent task storage.
* Tasks were read from the file at startup using the openTaskFile function and written back upon modifications.

**Graphical Enhancements:**

* Custom icons and images were added using the PhotoImage widget to improve visual appeal.
* Colors and fonts were carefully selected to ensure a modern and user-friendly interface.

**Event Handling**

* Tkinter's event-driven programming model was utilized to handle user actions (e.g., button clicks, listbox selections).
* All user interactions triggered their respective functions for seamless functionality.

**Testing and Debugging:**

The application was tested rigorously for edge cases such as:

Empty task input.

Attempting to update or delete without selecting a task.

Handling file read/write errors.

**Final Integration:**

All components were integrated into a cohesive application.

The main application window, login functionality, task management, and file handling were tested together to ensure smooth operation.

**Deployment:**

The application was packaged with necessary resource files (images and tasklist.txt).

Platform-independent features were verified to ensure compatibility across operating systems.

By following this implementation process, the **To-Do List Application** was successfully developed to provide users with an efficient task management tool.

**Result**

The To-Do List Application was successfully developed and tested. The results of the project are detailed below, based on functionality, usability, and performance:

**User Authentication:**

- The login feature was successfully implemented with password protection.

- Upon entering the correct password ("Baloch"), the user gains access to the main window of the application. If the password is incorrect, an error message is displayed. This ensures secure access to the To-Do list functionality.

**Task Management Features:**

Add Task:

The task input functionality worked as expected. Users could add tasks by typing in the input field and clicking the "ADD" button. The tasks were saved to a text file (tasklist.txt) for persistent storage and were displayed in the listbox.

Delete Task:

Users were able to delete selected tasks from the listbox and the text file. The delete operation worked without issues, ensuring that both the UI and storage were updated accordingly.

Update Task:

The task update functionality was implemented and functional. When a user selects a task from the listbox and clicks the "UPDATE" button, they are prompted to enter the new task details. The listbox and tasklist.txt file were updated with the new information, ensuring that the changes were saved correctly.

Mark Task as Important:

The functionality to mark tasks as important using a star ("⭐") symbol was successfully implemented. Users could toggle the "important" status of tasks, and the changes were reflected both in the listbox and the text file. The star symbol was properly added or removed, depending on the current state.

**UI Design and Usability:**

The application featured a clean and organized user interface. The main window displayed tasks in a listbox with a scrollbar for navigation, and buttons for adding, deleting, updating, and marking tasks as important were easily accessible.

Custom icons and images were used to enhance the appearance of the application. Visual elements like icons for updating, deleting, and marking tasks as important were intuitively placed to guide the user experience.

Task input was facilitated by a Text widget, which provided ample space for multi-line input. The design was optimized for ease of use, with the task list displayed clearly in a scrollable list.

**Data Persistence:**

The task list was successfully stored in the tasklist.txt file. When the application was reopened, the tasks were loaded from this file and displayed in the listbox, preserving the state of the to-do list between sessions.

Both task addition and task modification (update and important) operations were reflected in the text file, ensuring that user data was consistently saved and maintained.

**Error Handling:**

The application handled errors efficiently. For example, when trying to access an empty task list file, the application created a new file automatically.

If the user attempted to perform an action (like updating or deleting) without selecting a task, the application did not crash and was designed to handle such scenarios gracefully.

**Performance:**

- The performance of the application was smooth and responsive. The operations such as adding, updating, and deleting tasks were performed instantly without noticeable lag or delays.

- The application was able to handle a reasonable number of tasks in the list without performance degradation.

**Challenges**

### ****User Authentication Security****

* The login system relies on a hard-coded password ("Baloch"), which can be easily bypassed if the code is accessed. A more secure authentication system, such as using hashed passwords or connecting to a database, should be implemented.

### 2. ****Error Handling for File Operations****

* The program handles file operations with limited error checking. If the tasklist.txt file is missing, the program attempts to create a new file but doesn’t check for other potential errors such as file permission issues or data corruption.

### 3. ****Data Persistence Issues****

* While the tasks are saved to a file, there are no backups or recovery mechanisms in place. If the file becomes corrupted or lost, the task list would be lost without any chance of recovery.

### 4. ****Task Updates Not Synchronized with Task List****

* When a task is updated, the updated task is written back to the file, but there is a potential mismatch between the visual listbox and the task list in the program. The update process could be streamlined for better synchronization between the listbox and the task\_list array.

### 5. ****File Locking for Concurrent Access****

* The application does not account for concurrent access to the tasklist.txt file. If multiple instances of the program were to run, it might cause inconsistencies in the data being written to the file.

### 6. ****Task Importance Feature Potential Bugs****

* The "Important" feature marks tasks with a "⭐" symbol. While this works on the UI, the method for updating the task list in the file could result in unexpected behavior if a task is marked as important more than once or removed improperly.

### 7. ****Limited Task Filtering and Sorting****

* The application only supports adding, deleting, updating, and marking tasks as important. It lacks advanced features like filtering tasks by importance or due date, or sorting tasks based on priority.

### 8. ****UI Responsiveness and Accessibility****

* The current user interface uses fixed dimensions and does not adapt well to different screen sizes. There are also no accessibility features, such as keyboard navigation or high-contrast color schemes for users with visual impairments.

### 9. ****Scalability Issues****

* The application may not scale well if the number of tasks grows significantly. As the number of tasks increases, the user interface could become sluggish, especially when handling large amounts of data in the listbox or file.

### 10. ****Task Completion Status****

* The system does not support marking tasks as completed, which would be a useful addition to allow users to track their progress. Implementing a task completion feature could enhance the app's utility.

### 11. ****Limited Feedback for User Actions****

* While some operations like adding and deleting tasks are visually reflected in the listbox, there is no feedback provided for other actions like updating tasks or marking tasks as important. A notification or confirmation message would improve the user experience.

### 12. ****No Password Recovery System****

* The login screen relies on a single hardcoded password with no password recovery or reset option. This could be a limitation in terms of user experience and security.

These challenges offer insight into potential improvements, such as enhancing security, adding more features, improving the user interface, and addressing scalability concerns.

### Conclusion :

In conclusion, this project demonstrates the development of a simple yet functional To-Do List application using Python and the Tkinter library for the graphical user interface (GUI). The application allows users to efficiently manage tasks by providing features such as adding, updating, deleting, and marking tasks as important. These operations are saved to and loaded from a local text file, ensuring that user data persists between sessions.

Despite its simplicity, the project showcases the key aspects of software development such as file handling, GUI design, and user interaction. It also highlights the importance of task management tools in everyday life, as this app provides users with an organized and accessible way to keep track of their to-do items.

However, there are several areas for improvement, such as enhancing the security of the login system, refining task data integrity, and introducing more advanced features like due dates, task categorization, and better error handling. These enhancements would further elevate the app’s usability, performance, and scalability.

Overall, this To-Do List app serves as a great starting point for further development, and it can be expanded with additional features and optimizations to make it a more robust and user-friendly task management tool.

### References

1. Tkinter Documentation:

The official Tkinter documentation provides detailed information on various widgets, modules, and techniques. (link: <https://docs.python.org/3/library/tkinter.html)>

1. Youtube Tutorial Videos :

Lecture from youtube channel Apna College

1. Real Python Tkinter Guide:

Real Python provides an in-depth guide to Tkinter, including best practices, examples, and resources. (link: <https://docs.python.org/3/tutorial/inputoutput.html)>

1. ChatGPT:

You can ask ChatGPT for specific guidance on improving your To-Do-List App, such as optimizing code, implementing new features, or troubleshooting issues.