Project Report

Student Name:Ansin madhav **UID: 24MCA20193**
Branch: MCA General **Section/Group: 3B**
Semester: 1 st **Date of Performance: 29-Oct**
Subject Name: PYTHON PROGRAMMING LAB **Subject Code: 24CAH-606**

1. Aim of the project:

The aim of the project is to develop a simple and interactive to-do list application that allows users to efficiently manage their tasks. This program will enable users to:

- Add new tasks to keep track of their activities.
- View the current list of tasks with a clear indication of completed and pending items.
- Mark tasks as complete to stay organized and prioritize effectively.
- Delete tasks once they are finished or no longer relevant

2. Hardware and Software Requirements:

- Hardware Requirements:

1. Processor (CPU):

- Minimum: dual-core processor (e.g., Intel i3, AMD Ryzen 3).
- Recommended: Quad-core processor , especially when running multiple extensions or working on large projects.

2. Memory (RAM):

- Minimum: 4 GB of RAM.
- Recommended: 8 GB or more, especially if you are running multiple instances of Jupyter, using extensions, or working with large datasets or projects.

3. Storage:

- Minimum: 128 GB of storage. An SSD is preferable for faster access and better performance.
- Recommended: 256 GB SSD or more, providing sufficient space for your OS, Jupyter, Python libraries, and project files.

4. Graphics:

- Minimum: Integrated graphics will be sufficient for standard Python coding.
- Recommended: Dedicated graphics if you plan to work with GPU-intensive tasks, such as machine learning or complex visualizations.

5. Display:

- Minimum: A display with at least 1366x768 resolution.
- Recommended: Full HD (1920x1080) or higher for better clarity and screen real estate, especially when working with multiple code windows or side-by-side views.

- **Software Requirements**

  1. Operating System:

     - Windows: Windows 10 or later.
     - macOS: macOS 10.14 (Mojave) or later.
     - Linux: Any modern Linux distribution (e.g., Ubuntu 20.04 LTS, Fedora, etc.).

  2.Python Installation:

  3.Python Version: Python 3.6 or later. Download the latest version from the official Python website\ 4. Install Anaconda and Jupyter Notebook:

▪ Downlods and install Anaconda from https://repo.anaconda.com/archive/Anaconda3-2022.05-Windowsx86_64.exe.

▪ Open “Anaconda Prompt” by finding it in the windows (start) Menu. o Type the command in (python - - version) Anaconda was installed.

4. Start Jupyter Notebook: o Type the command in (Jupyter Notebook”) to Start Jupyter Notebook

3. Program Logic:

- Initialize:

  Start with an empty list to store tasks.

- Main Menu Loop:

  - Continuously display menu options:

    1. Add Task
    2. View Tasks
    3. Complete Task
    4. Delete Task
    5. Exit

- User Choice Handling:

  - Add Task: Prompt for a task description, then add it to the task list as "incomplete."
  - View Tasks: Display each task with its status (complete/incomplete).

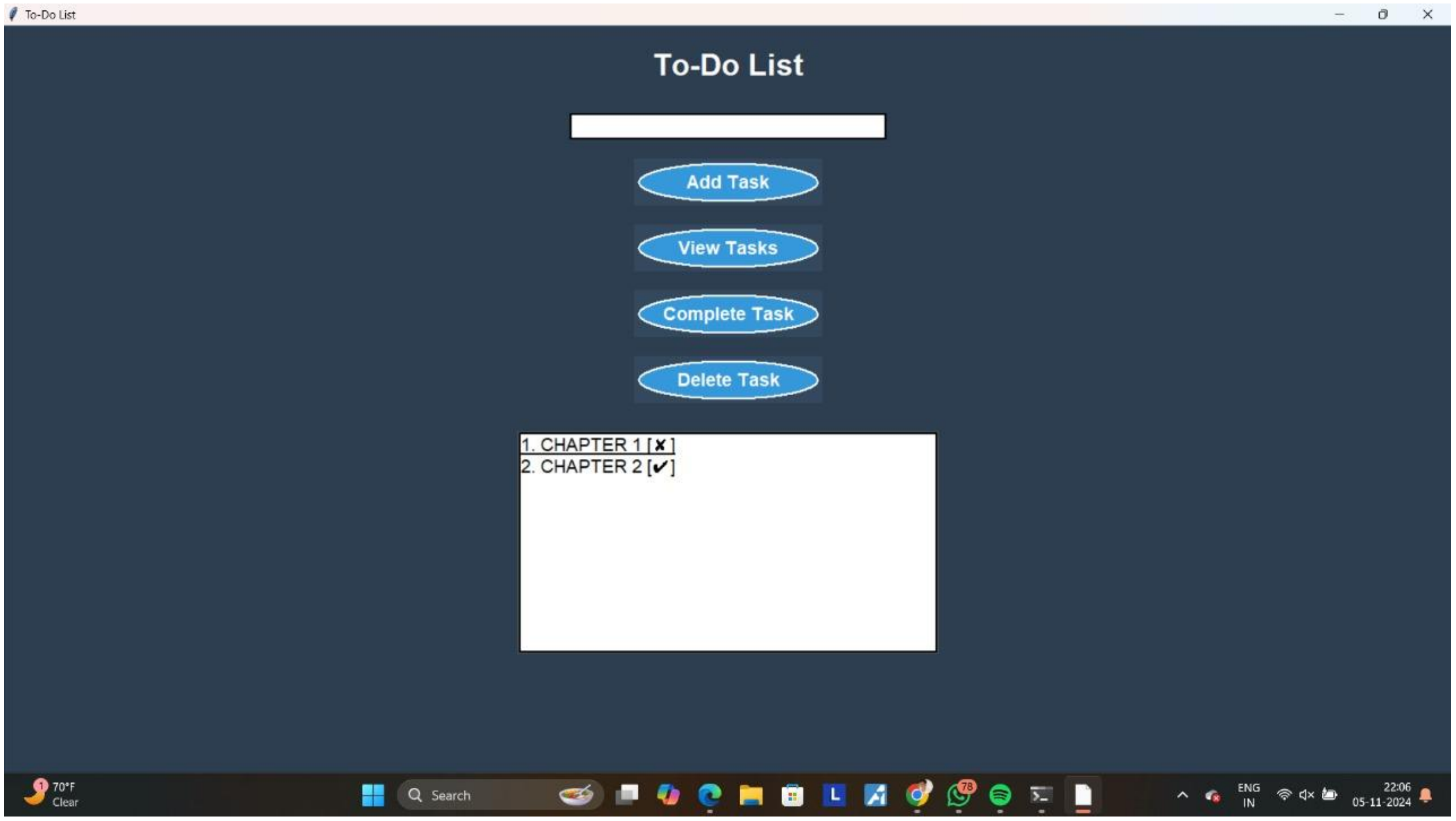
- Complete Task: Show tasks, prompt for task number, then mark the selected task as complete.
- Delete Task: Show tasks, prompt for task number, then delete the selected task.
- Exit: End the loop to close the program.

- Error Handling:
  - Check for invalid inputs (e.g., out-of-range task numbers) and notify the user.
- Loop Until Exit:
  - Return to the menu after each action, continuing until the user selects "Exit."

4. Code:

```
import tkinter as tk
from tkinter import messagebox

class TodoApp:
    def __init__(self, root):
        self.root = root
        self.root.title("To-Do List")
        self.root.geometry("400x500")
        self.root.config(bg="#2c3e50")

        self.tasks = []
        self.heading = tk.Label(self.root, text="To-Do List", font=("Helvetica", 24, "bold"), bg="#2c3e50", fg="white")
        self.heading.pack(pady=20)


        self.task_entry = tk.Entry(self.root, font=("Helvetica", 14), width=30, bd=2, relief="solid")
        self.task_entry.pack(pady=10)


        self.create_rounded_button(self.root, "Add Task", self.add_task)
        self.create_rounded_button(self.root, "View Tasks", self.view_tasks)
        self.create_rounded_button(self.root, "Complete Task", self.complete_task)
        self.create_rounded_button(self.root, "Delete Task", self.delete_task)


        self.task_display = tk.Listbox(self.root, font=("Helvetica", 14), height=10, width=40, bd=2, relief="solid",
selectmode=tk.SINGLE)
        self.task_display.pack(pady=20)

    def create_rounded_button(self, parent, text, command):
        """Create a rounded button using a canvas."""
        canvas = tk.Canvas(parent, width=200, height=50, bg="#34495e", highlightthickness=0)
        canvas.pack(pady=10)


        button_bg = canvas.create_oval(5, 5, 195, 45, fill="#3498db", outline="white", width=2)


        canvas.create_text(100, 25, text=text, fill="white", font=("Helvetica", 14, "bold"))


        canvas.tag_bind(button_bg, "<Button-1>", lambda e: command())

    def add_task(self):
        task = self.task_entry.get()
        if task:
            self.tasks.append({"task": task, "completed": False})
            self.task_entry.delete(0, tk.END)
            messagebox.showinfo("Task Added", f'Task "{task}" added successfully!')
        else:
            messagebox.showerror("Input Error", "Please enter a task.")

    def view_tasks(self):
        self.task_display.delete(0, tk.END)
        for i, task in enumerate(self.tasks, start=1):
            status = '✓' if task['completed'] else '✗'
            self.task_display.insert(tk.END, f"{i}. {task['task']} [{status}]")

    def complete_task(self):
        try:
            task_index = int(self.task_display.curselection()[0])
            if not self.tasks[task_index]['completed']:
                self.tasks[task_index]['completed'] = True
                self.view_tasks()
                messagebox.showinfo("Task Completed", f'Task "{self.tasks[task_index]["task"]}" marked as
complete!')
            else:
                messagebox.showinfo("Already Completed", "This task is already completed.")
        except IndexError:
            messagebox.showerror("Selection Error", "Please select a task to mark as complete.")

    def delete_task(self):
        try:
            task_index = int(self.task_display.curselection()[0])
            task = self.tasks.pop(task_index)
            self.view_tasks()
            messagebox.showinfo("Task Deleted", f'Task "{task["task"]}" deleted!')
        except IndexError:
            messagebox.showerror("Selection Error", "Please select a task to delete.")


if __name__ == "__main__":
    root = tk.Tk()
    app = TodoApp(root)
    root.mainloop()
```

5.Result:



```
1. CHAPTER 1 [✗]
2. CHAPTER 2 [✔]
```

6. Learning outcomes (What I have learnt):

- **Improved Coding Skills**: Learned to write organized code and use functions to make tasks easier to manage.
- **Control Flow**: Gained experience using loops and conditions to handle different options in the program smoothly.
- **Handling Errors**: Learned to check for mistakes in user input to make the program more reliable.
- **Data Management**: Practiced adding, updating, and deleting items in a list, so changes happen right away.
- **Problem-Solving**: Got better at breaking down complex problems into smaller steps to solve them effectively.
- **User-Friendly Design**: Learned to create simple menus and prompts to make the program easy to use.

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Worksheet		8 Marks
2.	Viva		10 Marks
3.	Simulation		12 Marks
4.	Total		30 Marks

Teacher's Signature