Week 1

- Use ai to understand python and its capabilities
- Brainstorm python mini project ideas
- Develop Logic Diagram
- Complete/present mini project
- Decide whether to redirect or advance with current project

Python To-Do List

DAE

07/10/25

AJ

Algorithm: To-Do List Manager (Step-by-Step)

1. Start the program
→ Load previously saved tasks from the `tasks.json` file, if it exists.

2. Show the main menu
→ Display available options: Add, View, Remove, Mark Complete, Exit.

3. Get the user's menu choice
→ Ask the user to input a number (1–5).

4. Use conditionals to process the choice
→

- If **1**, prompt for task description and add it to the list
- If **2**, display all current tasks with status
- If **3**, ask for a task ID and remove it from the list
- If **4**, ask for a task ID and mark it as complete
- If **5**, save all tasks to file and exit
- If invalid, show an error and re-display the menu

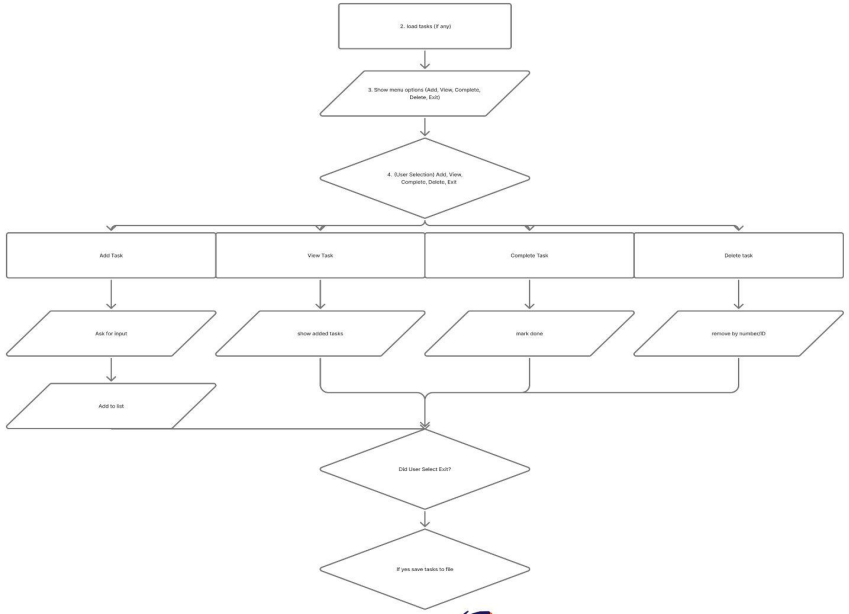
5. Repeat the loop until the user chooses to exit

6. On exit, save all tasks to the `tasks.json` file

7. End the program

Section 1

Logic Diagram



Why a To-Do List Manager?

Bullet Points:

- Helps users track daily responsibilities clearly and efficiently
- Teaches fundamental programming logic and data handling
- Can be expanded into a mobile or web app later
- Solves a real-world problem: disorganization and task overload
- Great foundation for mastering Python skills with practical value

Real-Life Use Case

Slide Title: *How This App Helps Daily Life*

Bullet Points:

- Track homework, personal goals, chores, habits, or deadlines
- Save your task list between uses — like a mini digital planner
- Adaptable: Can evolve into a web or mobile tool
- Great for people who prefer a **command-line workspace**
- Helps develop **self-discipline, consistency, and focus**

Core Python Features I Used

Bullet Points:

- **Functions** – Broke actions into reusable blocks (`add_task()`, `save_tasks()`, etc.)
- **Lists & Dictionaries** – Stored dynamic tasks and task details
- **Loops** – Powered the interactive menu and task processing
- **Conditionals** – Controlled logic based on user input and task states
- **File Handling** – Saved progress across sessions using `.json`
- **Try/Except** – Made the program resistant to invalid input

```python
def main():
    load_tasks()
    while True:
        print("\nTo-Do List Manager")
        print("1. View Tasks")
        print("2. Add Task")
        print("3. Remove Task")
        print("4. Mark Task Complete")
        print("5. Save & Exit")
```

```python
import json
import os

TASKS_FILE = "tasks.json"

tasks = []

def load_tasks():
    global tasks
    if os.path.exists(TASKS_FILE):
        with open(TASKS_FILE, "r") as f:
            tasks = json.load(f)
    else:
        tasks = []

def save_tasks():
    with open(TASKS_FILE, "w") as f:
        json.dump(tasks, f, indent=4)

def add_task():
    description = input("Enter task description: ").strip()
    if description:
        task = {
            "id": len(tasks) + 1,
            "description": description,
            "completed": False
        }
        tasks.append(task)
        print(f"Task added: {description}")
    else:
        print("Task description cannot be empty.")

def view_tasks():
    if not tasks:
        print("No tasks found.")
        return
    print("\nTasks:")
    for task in tasks:
        status = "✓" if task["completed"] else "✗"
        print(f"{task['id']}. [{status}] {task['description']}")

    try:
        task_id = int(input("Enter task ID to remove: "))
        for task in tasks:
            if task["id"] == task_id:
                tasks.remove(task)
                print(f"Removed task {task_id}")
                return
        print("Task ID not found.")
    except ValueError:
        print("Invalid input. please enter a number.")
```