

**The Superior University**

Operating Systems Lab – Project Documentation

# Project Title*:* Weakly Task Scheduler

# Group Members

# Daniyal Nasir ( BSEM-F20-171)

# Shafiq Haider (BSSEM-F23-071

# M Imran (BSSEM-F23-069)

# Asad Aslam (BSSEM-F23-082)

# GitHub Repository

**GitHub Repository Link:**

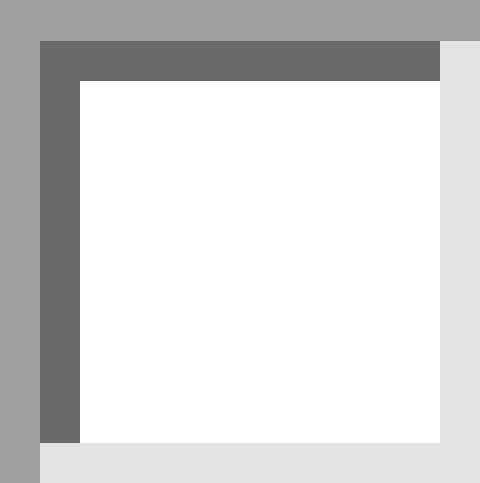
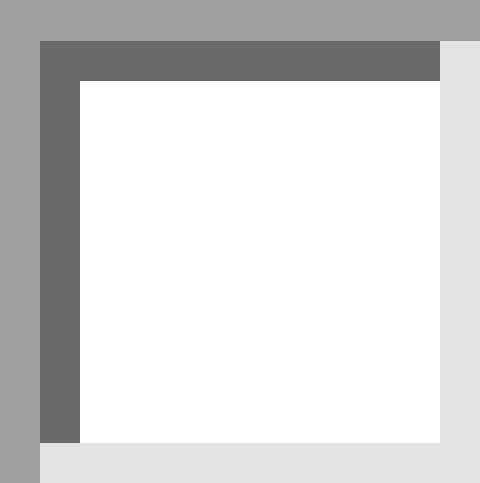
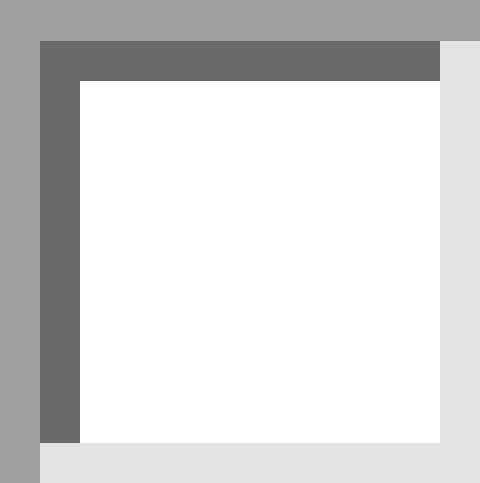
*https://github.com/AsadAslam967/Weekly-Task-scheduler-.git*

Your repository should include:

* ✔ Python .py file containing the scheduling algorithm
* ✔ README.md with basic project info and instructions
* ✔ This completed documentation file (.docx or .pdf)
* ✔ At least 2–3 output screenshots

# 🔧 Scheduling Algorithm Implemented

✅ Tick the scheduling algorithm your group implemented:

* ✅ FCFS (First Come First Serve)
*  SJF (Shortest Job First – Non-Preemptive)
*  SJF (Preemptive)
*  Round Robin

# Project Description

**What Problem Your Project Solves:**

The **Weekly Task Scheduler** solves the problem of disorganized weekly planning. It provides users with a structured interface to manage their tasks for each day of the week. Instead of relying on paper planners or forgetting daily tasks, users can add, view, and remove tasks digitally in an organized, simple format. It promotes better **time management**, **productivity**, and **task tracking** throughout the week.

**What Inputs Are Required:**

This project is not based on CPU scheduling concepts like burst time or arrival time. Instead, it uses customized inputs that are more suitable for a human task scheduler:

* **Day**: The day of the week (e.g., Monday, Tuesday).
* **Task**: A string describing the task (e.g., "Attend team meeting at 10 AM").
* **Index**: (Only for deletion) the number of the task to be removed from a specific day.

**What Outputs Are Generated:**

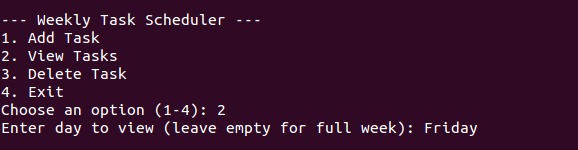
The application generates the following outputs:

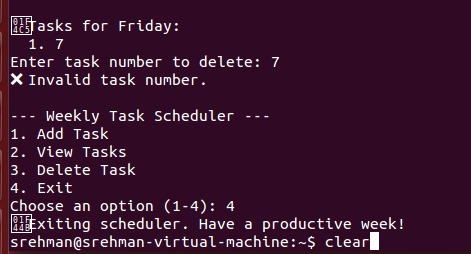
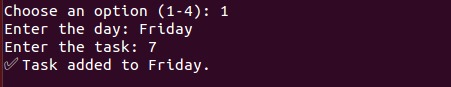
* **Task Confirmation**: Message confirming the task has been added to the selected day.
* **Weekly Schedule View**: Displays all tasks organized by day.
* **Day-Specific View**: Shows only the tasks scheduled for a specific day.
* **Task Deletion Feedback**: Confirmation or error messages when removing a task.
* **Error Messages**: For invalid input such as incorrect day names or invalid task numbers.

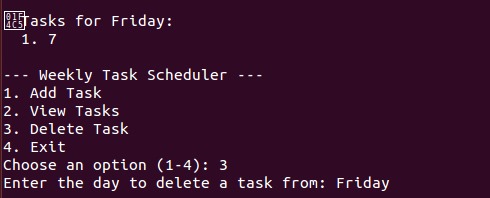
**How the Algorithm is Implemented:**

* A class named **WeekTaskScheduler** is used to maintain a dictionary of lists, where each key is a weekday, and the value is a list of tasks.
* The scheduler offers three main operations:
  1. **Add Task** (add\_task()): Appends a task to the list of a specified day.
  2. **View Task(s)** (view\_tasks()): Displays all tasks for a given day or the full week's schedule.
  3. **Delete Task** (remove\_task()): Removes a task by its index from a day’s task list.
* The main function (main()) provides a **menu-driven interface** allowing users to perform these actions in a loop until they choose to exit.
* The scheduler handles user input and provides feedback for each action, simulating a basic interactive task scheduling environment.

# Output Screenshots







# Code Structure & Explanation

**Functions or Classes Used:**

The code is organized using a single class and multiple functions:

* **Class: WeekTaskScheduler**  
  This class manages the core functionality of the scheduler. It contains a dictionary self.schedule where each key is a day of the week, and each value is a list of tasks for that day.
* **Methods within the class:**
  + add\_task(day, task): Adds a task to the list corresponding to a specified day.
  + view\_tasks(day=None): Displays all tasks for a specific day or, if no day is specified, the full weekly schedule.
  + remove\_task(day, index): Removes a task from the list of a specified day based on its index.
* **Function: main()**  
  This function provides a menu-based interface using a loop to allow users to:
  + Add a task
  + View tasks
  + Delete a task
  + Exit the program  
    It takes user input and calls the appropriate method from the WeekTaskScheduler class.

**Core Logic of the Scheduling Algorithm:**

Although this project is not a CPU scheduling algorithm in the traditional sense, it follows a structure similar to First Come, First Serve (FCFS):

* Tasks are stored and displayed in the order they are added.
* Each day maintains its own task queue (a list), and new tasks are appended to the end.
* Tasks are removed by selecting their index from the list.

The control flow is handled through user inputs via a loop and conditional statements that direct user interaction with the program.

**Any External Libraries Used:**

No external libraries are used in this project.

The code relies entirely on Python’s standard input/output and built-in data structures such as dictionaries and lists, making it lightweight and easy to run in any Python environment.

# Performance Metrics

# Note: This project implements a custom task scheduling system based on user-defined input rather than CPU-level scheduling algorithms, so traditional metrics do not apply.

**Challenges Faced**

* **Logic Bugs in Task Scheduling**  
  *Challenge*: The main issue was getting the tasks to schedule correctly based on priority and execution time, especially when dealing with overlapping tasks and waiting time.  
  *Solution*: We broke down the logic into smaller steps and used unit testing to identify where the scheduling process was going wrong. We also added print statements to track the task flow at each step, which helped us pinpoint and fix the logic error related to task queue management.
* **Incorrect Waiting Time Calculation**  
  *Challenge*: There was an issue with calculating the waiting time for tasks, which affected the performance metrics. Tasks were being executed in the wrong order, leading to incorrect waiting times and turnaround times.  
  *Solution*: We revisited the formulas for calculating waiting times and turnaround times, ensuring that the tasks were sorted and processed in the right order. After testing with different task combinations, we refined the algorithm to correctly handle these calculations.
* **Gantt Chart Representation**  
  *Challenge*: Generating an accurate Gantt chart that visually represented the task scheduling was harder than anticipated. The chart didn’t align well with the actual task execution timeline.  
  *Solution*: We utilized a library or tool that supported Gantt chart visualization and ensured that the start and end times for each task were accurately mapped to the chart. We also incorporated some trial-and-error to tweak the formatting for better clarity.