# University of Westminster

**Module Name:** Software Development 1

**Module leader**: Guhanathan Poravi

**Assignment type:** Individual coursework

**Group ID**: 06

**Submission date:** 31.03.2025

| Student Id | Uow ID | Student Name |
|---|---|---|
| 20242053 | w2152911 | S.A.D.D.N. Dissanayake |


School of Computing

Informatics Institute of Technology (IIT Campus)

# Table of Contents



# Table of figures

# [Stage 1] Pseudocode for Personal Task Manager

## Program description:

This code creates a basic Python command-line Personal Task Manager that lets users organize their tasks using a menu-driven interface. For task management, the application supports the fundamental CRUD (Create, Read, Update, Delete) operations. This is an explanation of how it works:

**1. Structure of Data:**

Each task is represented as a list with its name, description, priority, and due date, and the tasks are kept in a list called tasks.

**2. Functions:**

add_task(): Appends a new task to the tasks list after prompting the user to enter its name, description, priority, and due date.
view_tasks(): Shows every task that is presently listed in the tasks list. It notifies the user if there are no tasks.
update_task(): Selects an existing task by its index, enabling the user to update it. Which field to update—the name, description, priority, or due date—is up to the user.
delete_task(): Enables the user to delete a task by selecting it via its index. It removes the task from the tasks list.

**3. Interface for Users:**

A menu with options to add, view, update, delete, and exit tasks is displayed while the program runs in a loop. By inputting the appropriate number, the user can make choices.
To make sure the user enters correct task indices and options, input validation is incorporated.

**4. Handling Errors:**

Basic error handling for invalid inputs, such as non-integer task selection values or invalid task numbers, is built into the code.
All things considered, this code offers users a simple method of managing their tasks, making it simple to add, view, edit, and remove tasks as necessary.

## [Stage 1] Steps of the Program:

```
BEGIN
  Initialize tasks list as empty list  // List to store tasks

  FUNCTION add_task()  // Function to add a new task
    PROMPT user to enter task name
    PROMPT user to enter task description
    PROMPT user to enter task priority (high/medium/low)
    PROMPT user to enter task due date (YYYY-MM-DD)
    Add the task to the tasks list  // Store the task in the list
    PRINT "Task added successfully!"

  FUNCTION view_tasks()  // Function to view all tasks
    IF tasks list is empty THEN
      PRINT "No tasks available!"
      RETURN
    END IF

    PRINT "Tasks List:"
    FOR each task in tasks
      PRINT task details (name, description, priority, due date)
    END FOR

  FUNCTION update_task()  // Function to update an existing task
    IF tasks list is empty THEN
      PRINT "No tasks available to update!"
      RETURN
    END IF

    PROMPT user to enter task number to update (1-based index)
    IF the task number is invalid THEN
```

```
        PRINT "Invalid task number!"
        RETURN
    END IF

    PRINT "Select the field to update:"
    PRINT options (Name, Description, Priority, Due Date)

    PROMPT user to select a field to update
    IF the option is valid THEN
        PROMPT user to enter new value for the selected field
        Update the corresponding task field in the tasks list  // Update the task
        PRINT "Task updated successfully!"
    ELSE
        PRINT "Invalid option."
    END IF

FUNCTION delete_task()  // Function to delete an existing task
    IF tasks list is empty THEN
        PRINT "No tasks available to delete!"
        RETURN
    END IF

    PROMPT user to enter task number to delete (1-based index)
    IF the task number is invalid THEN
        PRINT "Invalid task number!"
        RETURN
    END IF

    DELETE the task from the tasks list  // Remove the task from the list
    PRINT "Task deleted successfully!"
```

```
MAIN  // Main program loop
    LOOP indefinitely
        PRINT "Task Manager"
        PRINT menu options (Add Task, View Tasks, Update Task, Delete Task, Exit)

        PROMPT user to choose an option
        IF option is "1" THEN
            Call add_task()
        ELSE IF option is "2" THEN
            Call view_tasks()
        ELSE IF option is "3" THEN
            Call update_task()
        ELSE IF option is "4" THEN
            Call delete_task()
        ELSE IF option is "5" THEN
            PRINT "Exiting Task Manager!"  // Exit message
            BREAK the loop
        ELSE
            PRINT "Invalid choice! Please try again!"  // Invalid option
        END IF
    END LOOP
END
```

## [Stage 1] Test cases table

| Test case No. | Input | Expected Outcome | Actual Outcome | Result |
|---|---|---|---|---|
| 1 | **(Adding a Task)**<br>Valid Input:<br>Given the user enters a task name, description, priority (high, medium, low), and a valid due date | **Then** the application should add the task and display "Task added successfully!" | As Expected, | Pass |
| 2 | **(Viewing Tasks)**<br>Viewing When No Tasks Exist:<br>Given there are no tasks in the list. | **Then** the application should display "No tasks available!" | As Expected, | Pass |
| 3 | **(Updating a Task)**<br>Updating When No Tasks Exist:<br>Given there are no tasks in the list. | **Then** the application should display "No tasks available to update!" | As Expected, | Pass |
| 4 | **(Deleting a Task)**<br>Deleting When No Tasks Exist:<br>Given there are no tasks in the list. | **Then** the application should display "No tasks available to delete!" | As Expected, | Pass |
| 5 | **(Invalid Input Handling)**<br>Entering a Non-Numeric Task Number:<br>Given the user enters letters or special characters instead of a number when asked for a task index. | **Then** the application should display "Invalid input! Please enter a number." | As Expected, | Pass |

# [Stage 1] Test cases and their outcomes.

## Test case 1 outcomes:

Task Addition Test: Verifies that the application can correctly accept and store a new task with all required fields (name, description, priority, due date) and confirm with a success message.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS                                   Python

PS D:\!IIT Work\!Course Work> & C:/Users/USER/AppData/Local/Programs/Python/Python313/python.exe "d:/!IIT Work/!Course Work/1st Year/Semester 01/Software Development 1 4C
OSC006C/CW 1/Stage 1 - List-Based Design and Basic CRUD Operations/Personal_Task_Manager.py"

Task Manager
1. Add Task
2. View Tasks
3. Update Task
4. Delete Task
5. Exit
Choose an option: 1
Enter task name: Do SD1 Course Work
Enter task description: Details on the BB
Enter task priority (high/medium/low): high
Enter due date (YYYY-MM-DD): 2025/03/31
Task added successfully!
```

*Figure 1: Case 1 outcomes [stage 1]*

## Test case 2 outcomes:

**Empty Task List View Test**: Ensures the application properly handles the case when a user tries to view tasks but none exist, displaying an appropriate message instead of showing empty data.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS                                   Python

PS D:\!IIT Work\!Course Work> & C:/Users/USER/AppData/Local/Programs/Python/Python313/python.exe "d:/!IIT Work/!Course Work/1st Year/Semester 01/Software Development 1 4C
OSC006C/CW 1/Stage 1 - List-Based Design and Basic CRUD Operations/Personal_Task_Manager.py"

Task Manager
1. Add Task
2. View Tasks
3. Update Task
4. Delete Task
5. Exit
Choose an option: 2
No tasks available!
```

*Figure 2: Case 2 outcomes [stage 1]*

## Test case 3 outcomes:

**Empty Task List Update Test**: Confirms the application correctly responds when a user attempts to update a task when no tasks exist, preventing unnecessary user input and providing clear feedback.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS                                   Python

PS D:\!IIT Work\!Course Work> & C:/Users/USER/AppData/Local/Programs/Python/Python313/python.exe "d:/!IIT Work/!Course Work/1st Year/Semester 01/Software Development 1 4C
OSC006C/CW 1/Stage 1 - List-Based Design and Basic CRUD Operations/Personal_Task_Manager.py"

Task Manager
1. Add Task
2. View Tasks
3. Update Task
4. Delete Task
5. Exit
Choose an option: 3
No tasks available to update!
```

*Figure 3: Case 3 outcomes [stage 1]*

## Test case 4 outcomes:

**Empty Task List Delete Test**: Validates that the application handles attempts to delete tasks from an empty list appropriately, preventing potential errors and informing the user clearly.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    COMMENTS                    Python

PS D:\!IIT Work\!Course Work> & C:/Users/USER/AppData/Local/Programs/Python/Python313/python.exe "d:/!IIT Work/!Course Work/1st Year/Semester 01/Software Development 1 4C
OSC006C/CW 1/Stage 1 - List-Based Design and Basic CRUD Operations/Personal_Task_Manager.py"

Task Manager
1. Add Task
2. View Tasks
3. Update Task
4. Delete Task
5. Exit
Choose an option: 4
No tasks available to delete!
```

*Figure 4: Case 4 outcomes [stage 1]*

## Test case 5 outcomes:

**Non-Numeric Input Validation Test**: Tests the application's error handling when users provide invalid input (letters or special characters) for numerical fields, ensuring the program remains stable and provides helpful error messages.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    COMMENTS                    Python

Enter due date (YYYY-MM-DD): 2025/03/31
Task added successfully!


Task Manager
1. Add Task
2. View Tasks
3. Update Task
4. Delete Task
5. Exit
Choose an option: 3
Enter task number to update (1-based index): test
Invalid input! Please enter a number.
```

*Figure 5: Case 5 outcomes [stage 1]*

## [Stage 2] Pseudocode for Text File Handling for Task Persistence

### Program description:

File handling has been added to this updated Personal Task Manager, enabling tasks to be saved to a text file and retrieved upon program restart. Data persistence across sessions is ensured by automatically loading tasks from the file at startup. Task data is now updated as each CRUD operation adding, viewing, updating, and deleting interacts with the file. Because of these improvements, the task manager is now more dependable, intuitive, and effective, enabling users to manage their tasks without losing progress.

Additional file handling other than the stage 1:

- load_tasks_from_file(): Loads tasks from the file into memory at the start of the program.
- save_tasks_to_file(): Writes all tasks from memory to the file whenever a task is added, updated, or deleted.

## [Stage 2] Steps of the Program:

```
BEGIN TaskManager

    // Define the file to store tasks
    SET task_file TO "tasks.txt"

    // Initialize an empty list to store tasks in memory
    DECLARE tasks AS LIST

    // Function to load tasks from file
    FUNCTION load_tasks_from_file()
        // Use global tasks list
        SET global tasks
        TRY
            OPEN task_file FOR READING AS file
            FOR EACH line IN file
                // Split task details stored in the file
                SET task_details TO line.strip().split(" | ")
                IF LENGTH(task_details) == 4 THEN
                    // Add valid task details to the tasks list
                    APPEND task_details TO tasks
                END IF
            END FOR
        EXCEPT FileNotFoundError
            // If file does not exist, do nothing
            PASS
        END TRY
    END FUNCTION
```

```
// Function to save tasks to file
FUNCTION save_tasks_to_file()
    OPEN task_file FOR WRITING AS file
    FOR EACH task IN tasks
        // Write each task in a formatted manner
        WRITE " | ".join(task) TO file
    END FOR
END FUNCTION

// Function to add a task
FUNCTION add_task()
    // Get task details from user
    PROMPT "Enter task name: " AND STORE input AS task_name
    PROMPT "Enter task description: " AND STORE input AS description
    PROMPT "Enter task priority (high/medium/low): " AND STORE input AS priority
    PROMPT "Enter due date (YYYY-MM-DD): " AND STORE input AS due_date

    // Add task to the list
    APPEND [task_name, description, priority, due_date] TO tasks
    CALL save_tasks_to_file() // Save tasks to file for persistence
    PRINT "Task added successfully!"
END FUNCTION

// Function to view all tasks
FUNCTION view_tasks()
    IF tasks IS EMPTY THEN
        PRINT "No tasks available!"
        RETURN
    END IF
```

```
    PRINT "Tasks List:"
    FOR EACH task IN tasks WITH INDEX index
        PRINT "Task " + (index + 1) + ":"
        PRINT " Name: " + task[0]
        PRINT " Description: " + task[1]
        PRINT " Priority: " + task[2]
        PRINT " Due Date: " + task[3]
    END FOR
END FUNCTION

// Function to update a task
FUNCTION update_task()
    IF tasks IS EMPTY THEN
        PRINT "No tasks available to update!"
        RETURN
    END IF

    PROMPT "Enter task number to update (1-based index): " AND STORE input AS
task_index
    task_index = task_index - 1 // Convert to 0-based index

    IF task_index IS NOT IN RANGE(0, LENGTH(tasks)) THEN
        PRINT "Invalid task number!"
        RETURN
    END IF

    PRINT "Select the field to update:"
    PRINT "1. Name"
    PRINT "2. Description"
    PRINT "3. Priority"
    PRINT "4. Due Date"
```

```
    PROMPT "Enter the corresponding number: " AND STORE input AS criteria

    IF criteria IN ['1', '2', '3', '4'] THEN
        PROMPT "Enter the new value: " AND STORE input AS modified_value
        tasks[task_index][criteria - 1] = modified_value // Update task
        CALL save_tasks_to_file() // Save updated tasks
        PRINT "Task updated successfully!"
    ELSE
        PRINT "Invalid option."
    END IF
END FUNCTION

// Function to delete a task
FUNCTION delete_task()
    IF tasks IS EMPTY THEN
        PRINT "No tasks available to delete!"
        RETURN
    END IF

    PROMPT "Enter task number to delete (1-based index): " AND STORE input AS
task_index
    task_index = task_index - 1 // Convert to 0-based index

    IF task_index IS NOT IN RANGE(0, LENGTH(tasks)) THEN
        PRINT "Invalid task number!"
        RETURN
    END IF

    REMOVE tasks[task_index] // Remove task from list
    CALL save_tasks_to_file() // Save updated task list
```

```
    PRINT "Task deleted successfully!"
END FUNCTION

// Main execution starts here
CALL load_tasks_from_file() // Load existing tasks before showing menu

WHILE TRUE DO
    PRINT "Task Manager"
    PRINT "1. Add Task"
    PRINT "2. View Tasks"
    PRINT "3. Update Task"
    PRINT "4. Delete Task"
    PRINT "5. Exit"

    PROMPT "Choose an option: " AND STORE input AS choice

    IF choice == '1' THEN
        CALL add_task() // Call function to add task
    ELSE IF choice == '2' THEN
        CALL view_tasks() // Call function to view tasks
    ELSE IF choice == '3' THEN
        CALL update_task() // Call function to update task
    ELSE IF choice == '4'
```

## [Stage 2] Test cases table

| Test case No. | Input | Expected Outcome | Actual Outcome | Result |
|---|---|---|---|---|
| 1 | **(Task Deletion)** Valid Task Deletion: Given the user enters a valid task number. | Then the application should delete the task and display "Task deleted successfully!" | As Expected, | Pass |
| 2 | **(File Handling - Non-Existent File)** File Not Found: Given the program starts but tasks.txt does not exist. | Then the program should not crash, handle the missing file gracefully without errors, and display "No tasks available!" when viewing tasks. | As Expected, | Pass |
| 3 | **(File Handling - Persistence)** File Write on Task Addition: Given a user adds a new task using the program. | Then the task should be written to tasks.txt in the correct format for future retrieval. | As Expected, | Pass |
| 4 | **(File Handling - Task Modifications)** File Update on Task Modification: Given a user updates or deletes an existing task. | Then the changes should be properly reflected in tasks.txt, maintaining data consistency between program runs. | As Expected, | Pass |

# [Stage 2] Test cases and their outcomes.

## Test case 1 outcomes:

Verify basic task deletion functionality by creating a task, then deleting it using its displayed index, and confirming both the success message and that the task is no longer listed when viewing tasks.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS                                   Python

Enter due date (YYYY-MM-DD): 2025/03/31
Task added successfully!


Task Manager
1. Add Task
2. View Tasks
3. Update Task
4. Delete Task
5. Exit
Choose an option: 4
Enter task number to delete (1-based index): 1
Task deleted successfully!
```

*Figure 6: Case 1 outcomes [stage 2]*

## Test case 2 outcomes:

Delete or rename any existing tasks.txt file, start the application, attempt to view tasks, and verify the application launches without errors and correctly displays the "No tasks available!" message.

```python
# Function to load tasks from file
def load_tasks_from_file():
    """Loads tasks from the file into the tasks list."""
    global tasks  # Using the global tasks list
    try:
        with open(task_file, "r") as file:
            for line in file:
                task_details = line.strip().split(" | ")  # Splitting task details stored in the file
                if len(task_details) == 4:  # Ensuring the correct format
                    tasks.append(task_details)  # Adding to the list
    except FileNotFoundError:
        # If file does not exist, no tasks are loaded
        # This handles requirement #5 - File Not Found scenario
        pass

```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS                                   Python

PS D:\!IIT Work\!Course Work> & C:/Users/USER/AppData/Local/Programs/Python/Python313/python.exe "d:/!IIT Work/!Course Work/1st Year/Semester 01/Software Development 1 4C
OSC006C/CW 1/Stage 2 -  Text File Handling for Task Persistence/Personal_Task_Manager_file_handling.py"

Task Manager
1. Add Task
2. View Tasks
3. Update Task
4. Delete Task
5. Exit
Choose an option: 2
No tasks available!
```

*Figure 7: Case 2 outcomes [stage 2]*

## Test case 3 outcomes:

Add a task with unique identifiable details, exit the application completely, restart it, and verify the previously added task appears correctly when viewing tasks, confirming data persistence.

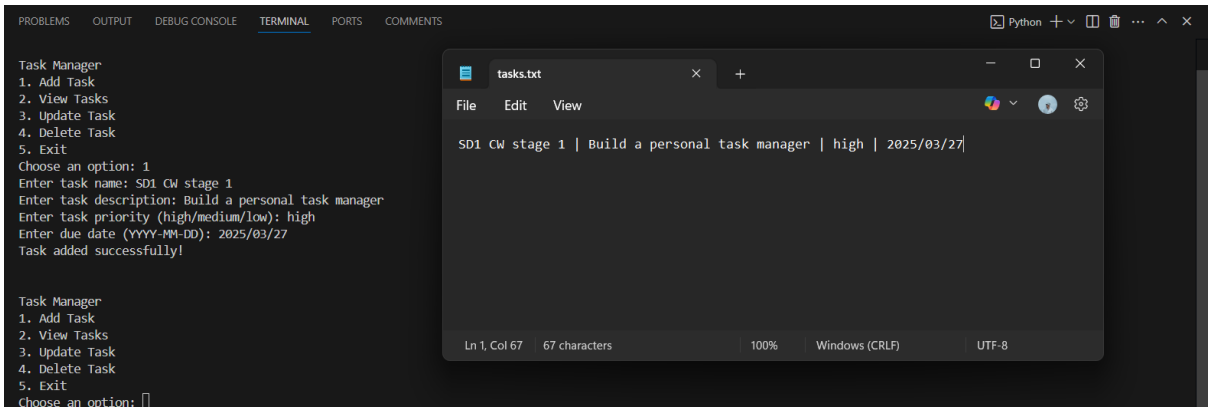


*Figure 8: Case 3 outcomes [stage 2]*

## Test case 4 outcomes:

Add a task, modify one of its fields, verify the change in the application, then exit and restart the application to confirm the modification persists across program sessions.

Before:

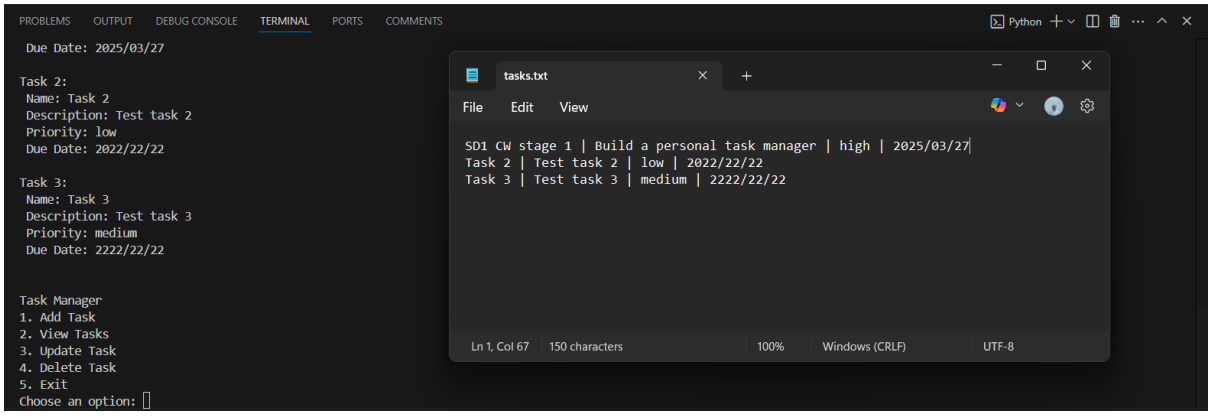


*Figure 9: Case 4 outcomes [stage 2]*

After:

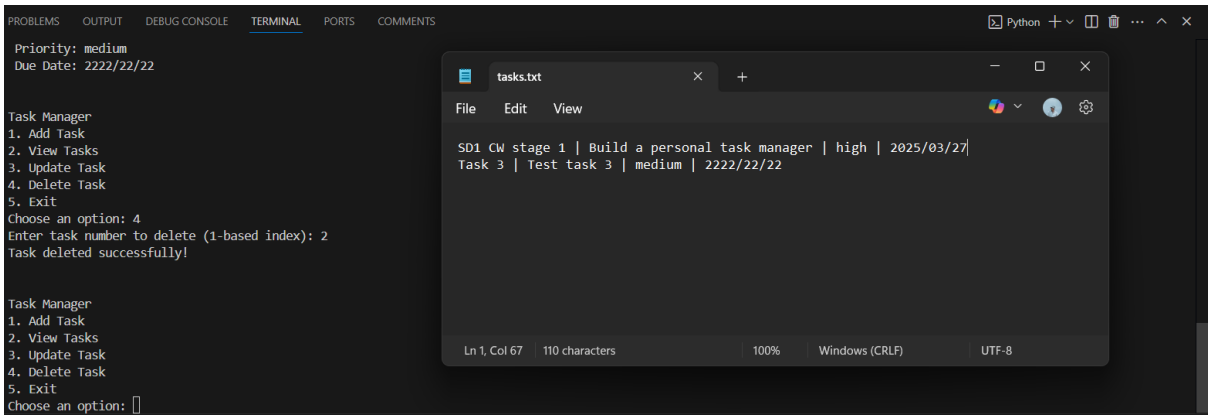


*Figure 10: Case 4 outcomes [stage 2]*