|  |  |
| --- | --- |
| **4COSC001W: Software Development I – Coursework specification (2024/25)** | |
| Module leader | Guhanathan Poravi (P.Guganathan@westminster.ac.uk) |
| Weighting: | 50% |
| Qualifying mark: | 30% |
| Description: | Coursework |
| Learning Outcomes Covered in this Assignment: | The coursework rationale is:  LO1 Analyse specific problems and design their solutions by applying appropriate algorithmic techniques;  LO2 Apply programming concepts to implement solutions in the taught programming language;  LO3 Implement and manipulate simple data structures;  LO4 Use an integrated development environment to create programs to satisfy a simple specification. |
| Handed Out: | Tuesday 19th Feb 2025 |
| Due Date: | **Coursework Deadline**:  Stage 01 and 02 – Thursday 27th Mar 2025 **BEFORE** 1:00 pm  Stage 03 and 04 – Monday 21st Apr 2025  **BEFORE** 1:00pm |
| Expected deliverables: | 1. Submit your Python program code    * **Important:** Submit **a single** zipped file containing design, code and test result using the name convention: “student\_id.zip”, e.g. w1234567.zip    * **DO NOT** submit your code as a word, notepad or PDF document. 2. Viva will be scheduled between 21st Apr 2054 to 11th May 2025 (schedules are tentative subject to change without prior notice) |
| Method of Submission: | Submitted online via Blackboard |
| Type of Feedback and Due Date: | Feedback during viva and written feedback and marks by 1st of Jun 2025 |

**Assessment regulations**

Refer to the following for clarification on what constitutes plagiarism, collusion and penalties for late submissions.

This is an individual coursework. You should not share your coursework or parts of your coursework with another student as this can cause you both to receive an allegation of collusion:

<https://www.westminster.ac.uk/current-students/guides-and-policies/academic-matters/academic-misconduct/collusion>Clarification on what constitutes plagiarism: <https://www.westminster.ac.uk/current-students/guides-and-policies/academic-matters/academic-misconduct/plagiarism>

**Penalty for Late Submission**

If you submit your coursework late but within 24 hours or one working day of the specified deadline, 10 marks will be deducted from the final mark, as a penalty for late submission, except for work which obtains a mark in the range 40 – 49%, in which case the mark will be capped at the pass mark (40%). If you submit your coursework more than 24 hours or more than one working day after the specified deadline you will be given a mark of zero for the work in question unless a claim of Mitigating Circumstances has been submitted and accepted as valid.

It is recognised that on occasion, illness or a personal crisis can mean that you fail to submit a piece of work on time. In such cases you must inform the Campus Office in writing on a mitigating circumstances form, giving the reason for your late or non-submission. You must provide relevant documentary evidence with the form. This information will be reported to the relevant Assessment Board that will decide whether the mark of zero shall stand. For more detailed information regarding University Assessment Regulations, please refer to the following website[: **http://www.westminster.ac.uk/study/current-students/resources/academic-regulations**](http://www.westminster.ac.uk/study/current-students/resources/academic-regulation)

**Coursework: Personal Task Manager (Using Lists, Dictionaries, JSON, and Tkinter with Classes and Object)**

**Overview:** This assignment involves developing a Personal Task Manager in Python. The project is structured into stages, beginning with basic list operations and file handling, progressing to JSON-based persistence, and concluding with a GUI built with Tkinter using Classes and Objects principles. Each stage has a specific submission deadline.

**Note:** You may use tools like ChatGPT to assist you with this assignment. However, you cannot copy and paste responses directly from ChatGPT without modification. Make sure to understand the concepts and write the code in your own words to avoid plagiarism.

**What is Academic Misconduct?** Academic misconduct includes any dishonest behavior that gives an unfair advantage in an academic setting. This includes plagiarism, cheating, fabrication, getting help from others on individual assignments, and facilitating academic dishonesty. Plagiarism specifically involves using someone else's work or ideas without proper attribution, presenting them as your own. Since this is an individual assignment, getting help from friends to complete it also constitutes academic misconduct. To avoid academic misconduct, always ensure that your work is original and follows the guidelines provided. If not, severe penalties may be applied.

**Bonus Marks: LinkedIn Learning Python Courses**

**Total Bonus Marks: Up to 10**

**1. Course Completion (Up to 10 Marks)**

* **Complete Up to Four Python Courses on LinkedIn Learning**
  + **Eligibility Period:** 24th Feb 2025 to 30th Apr 2025
  + **Requirements:**
    - Courses must be related to Python programming.
    - Provide links to certificates of successful completion ensure your name in the certificate is same as name in the SRSWeb UoW.
    - Each completed course earns **2.5 bonus marks**.
    - Maximum bonus marks achievable: **10 marks**.

| **Criteria** | **Marks** |
| --- | --- |
| - Completion of one course | 2.5 |
| - Completion of two courses | 5.0 |
| - Completion of three courses | 7.5 |
| - Completion of four courses | 10.0 |

**2. Submission Guidelines**

* **Proof of Completion**
  + Submit links or screenshots of certificates indicating successful completion of each course.
  + Ensure that the completion dates fall within the eligibility period.
* **Relevance of Courses**
  + Courses should be directly related to Python programming, such as:
    - Python fundamentals
    - Advanced Python techniques
    - Python for data analysis
    - Python GUI development

**Stage 1: List-Based Design and Basic CRUD Operations (Weeks 1-7)**

* **Objectives:**
  + Design the application using lists to store tasks, including details like task name, description, priority, and due date.
  + Implement basic CRUD (Create, Read, Update, Delete) operations with list manipulations only.
* **Deliverables:**
  + Pseudocode or Flowcharts for CRUD operations.
  + Python Code for CRUD functionalities using lists.
* **Submission Deadline:** Monday, 10th Mar 2025, at 1:00 PM.

**Code Template:**

# List to store tasks

tasks = []

# Functions for CRUD operations

def add\_task():

pass

def view\_tasks():

pass

def update\_task():

pass

def delete\_task():

pass

if \_\_name\_\_ == "\_\_main\_\_":

# Main function calls to test CRUD

pass

**Stage 2: Text File Handling for Task Persistence (Weeks 8-9)**

* **Objectives:**
  + Expand the application to save and load tasks from a text file, ensuring persistence between runs.
  + Use lists to store task details and manage file I/O operations.
* **Deliverables:**
  + Updated Python Code for reading from and writing to a text file.
  + Test Cases for file handling operations.
* **Submission Deadline:** Thursday, 27th Mar 2025, at 1:00 PM. (Summative)

**Code Template:**

# List to store tasks

tasks = []

# Functions for CRUD operations

def add\_task():

pass

def view\_tasks():

pass

def update\_task():

pass

def delete\_task():

pass

# File handling functions for saving and loading tasks

def load\_tasks\_from\_file():

pass

def save\_tasks\_to\_file():

pass

if \_\_name\_\_ == "\_\_main\_\_":

# Main function calls to test CRUD

# Integrate file handling with CRUD functions

pass

**Stage 3: Using Dictionaries and JSON File Handling (Weeks 10-11)**

* **Objectives:**
  + Transition from list-based storage to dictionaries, using JSON for structured data persistence.
  + Implement loading and saving tasks to a JSON file.
* **Deliverables:**
  + Python Code with dictionary-based task management and JSON handling.
  + Sample JSON File (see below for format).
  + Test Cases validating JSON file operations.
* **Submission Deadline:** Thursday, 10th Apr 2025, at 1:00 PM.

**Sample JSON File:**

[

{"name": "Pay bills", "description": "Electricity and water bills", "priority": "High", "due\_date": "2024-10-15"},

{"name": "Buy groceries", "description": "Milk, Bread, Eggs", "priority": "Medium", "due\_date": "2024-10-16"}

]

**Code Template:**

import json

# List of dictionaries for tasks

tasks = []

# Functions for CRUD operations

def add\_task():

pass

def view\_tasks():

pass

def update\_task():

pass

def delete\_task():

pass

# JSON file handling functions

def load\_tasks\_from\_json():

pass

def save\_tasks\_to\_json():

pass

if \_\_name\_\_ == "\_\_main\_\_":

# Main function calls to test CRUD

# CRUD functions to work with dictionaries and JSON

pass

**Stage 4: Tkinter GUI for Viewing, Searching, and Sorting Tasks (Weeks 12-13)**

* **Sample GUI Layout:**

Below is an example of how the GUI should look. The GUI should include the following components:

* + **Title Bar:** Displays the name of the application (e.g., 'Personal Task Manager').
  + **Search and Filter Section:** Entry fields and dropdown menus to filter tasks by name, priority, or due date.
  + **Task Display Table:** A Treeview widget that displays tasks in a tabular format, including columns for name, description, priority, and due date. The columns should be clickable to allow sorting by each attribute.
  + **Buttons:** Buttons for actions such as 'Filter' to apply filters and 'Sort' for sorting tasks based on selected criteria.

**GUI Layout** **Explanation:**

* + The **Title Bar** at the top simply displays the name of the application.
  + The **Search and Filter Section** includes text entry fields and dropdowns for filtering tasks by their name, priority, or due date, making it easy for users to narrow down the list of tasks.
  + The **Task Display Table** is implemented using a Tkinter Treeview widget, displaying all tasks in a tabular format with attributes such as name, description, priority, and due date.
  + **Column Headers** are clickable, allowing users to sort the tasks in ascending or descending order based on the respective attribute.
  + **Buttons for Sorting** are provided below the task table to allow users to sort tasks by name, priority, or due date with a single click.

This layout should help guide the implementation and ensure a user-friendly interface.

* **Objectives:**
  + Implement a Tkinter GUI to display all tasks in a tabular format.
  + Enable users to filter tasks by criteria such as name, priority or due date.
  + Allow users to sort tasks by clicking on column headers (e.g., by name, priority, or due date).
* **Class Structure:**
  + **Task Class:** Represents a task, containing properties such as name, description, priority, and due date.
  + **TaskManager Class:** Handles loading tasks from a JSON file and manages sorting and filtering operations.
  + **Tkinter GUI:** Provides a tabular display of all tasks and includes functionalities for filtering and sorting based on user-selected criteria.
* **Deliverables:**
  + Python Code with Classes and Object and Tkinter GUI for viewing, searching, and sorting tasks.
  + Documentation on the GUI interface and instructions for using search and sort functionalities.
* **Submission Deadline:** Monday, 21st Apr 2025, at 1:00 PM. (Summative)

**Code Template:**

import json

import tkinter as tk

from tkinter import ttk

# Define the Task class to represent each task

class Task:

def \_\_init\_\_(self, name, description, priority, due\_date):

# Initialize task properties

pass

def to\_dict(self):

# Convert task properties to a dictionary for JSON serialization

pass

# Define the TaskManager class to handle task operations

class TaskManager:

def \_\_init\_\_(self, json\_file='tasks.json'):

# Initialize TaskManager with a list of tasks and load tasks from JSON

pass

def load\_tasks\_from\_json(self):

# Load tasks from a JSON file into the task list

pass

def get\_filtered\_tasks(self, name\_filter=None, priority\_filter=None, due\_date\_filter=None):

# Return tasks filtered by name, priority, or due date

pass

def sort\_tasks(self, sort\_key='name'):

# Sort tasks by the specified key (e.g., name, priority, due date)

pass

# Define the TaskManagerGUI class to create the Tkinter interface

class TaskManagerGUI:

def \_\_init\_\_(self, root):

# Initialize GUI components and set up the Tkinter window

pass

def setup\_gui(self):

# Create and place GUI components (labels, entry fields, buttons, table)

pass

def populate\_tree(self):

# Display tasks in a tabular format in the Treeview widget

pass

def apply\_filter(self):

# Apply filter criteria based on user input and refresh the task display

pass

def sort\_tasks(self, sort\_key):

# Sort tasks by a specific column and update the task display

pass

# Main program execution

if \_\_name\_\_ == "\_\_main\_\_":

# Create the Tkinter root window and run the GUI application

root = tk.Tk()

app = TaskManagerGUI(root)

root.mainloop()

**Class and Method Descriptions:**

1. **Task Class**
   * **Properties:**
     + name: The name of the task.
     + description: A description of the task.
     + priority: The priority level of the task (e.g., "High", "Medium", "Low").
     + due\_date: The due date of the task in "YYYY-MM-DD" format.
   * **Methods:**
     + \_\_init\_\_(self, name, description, priority, due\_date): Initializes task properties.
     + to\_dict(self): Converts the task into a dictionary format for JSON serialization.
2. **TaskManager Class**
   * **Properties:**
     + tasks: A list to hold multiple Task objects.
     + json\_file: The path to the JSON file where tasks are stored.
   * **Methods:**
     + \_\_init\_\_(self, json\_file='tasks.json'): Initializes the task manager and loads tasks from the specified JSON file.
     + load\_tasks\_from\_json(self): Loads tasks from a JSON file into the tasks list.
     + get\_filtered\_tasks(self, name\_filter=None, priority\_filter=None, due\_date\_filter=None): Returns tasks filtered by name, priority, or due date.
     + sort\_tasks(self, sort\_key='name'): Sorts the tasks list by a specified attribute (name, priority, or due date).
3. **TaskManagerGUI Class**
   * **Properties:**
     + task\_manager: An instance of the TaskManager class to manage task data.
     + root: The root Tkinter window.
     + tree: A Treeview widget for displaying tasks in a tabular format.
   * **Methods:**
     + \_\_init\_\_(self, root): Initializes the GUI, sets up the window, and calls setup\_gui.
     + setup\_gui(self): Creates and places GUI components like labels, entry fields, buttons, and the task table.
     + populate\_tree(self): Populates the Treeview with tasks from the task manager.
     + apply\_filter(self): Filters tasks based on user input and refreshes the Treeview display.
     + sort\_tasks(self, sort\_key): Sorts the tasks in the Treeview by the specified column and updates the display.

**Detailed Marking Guide for Personal Task Manager Assignment**

This marking guide provides a detailed breakdown of how marks are allocated for each stage of the Personal Task Manager assignment. Use this rubric to understand how each component contributes to your overall score.

**Viva Voce Examination Guidelines**

1. **Mandatory Viva Voce Duration**  
   Each coursework submission will be followed by a viva voce examination of **not less than 30 minutes**. This viva is compulsory and contributes to the assessment of your coursework.
2. **Performance-Based Assessment**  
   The marking criteria may be revised based on your performance during the viva. If you are unable to effectively respond to questions or complete requested tasks, marks may be deducted from your overall assignment grade.
3. **Timing and Schedule**  
   The viva and presentation will be strictly timed. Your module leader will share the schedule for these components after the assignment submission deadline. Please note that no extra time will be provided.
4. **No Second Attempt**  
   There is no opportunity for a second viva or presentation, regardless of the reasons for an unsatisfactory performance in the first attempt. Therefore, thorough preparation and punctual attendance are crucial.
5. **Consequences of Absence  
   If you do not attend the scheduled viva voce examination, you will receive zero marks for the whole coursework.**

**Stage 1: List-Based Design and Basic CRUD Operations**

**Total Marks: 25**

**1. Design Documentation (10 Marks)**

* **Pseudocode or Flowcharts for CRUD Operations**

| **Criteria** | **Marks** |
| --- | --- |
| **Add Task Function** | 2.5 |
| - Clear and logical steps outlined | 1.5 |
| - Includes input validation | 0.5 |
| - Shows data being added to the list | 0.5 |
| **View Tasks Function** | 2.5 |
| - Correctly iterates over the task list | 1.5 |
| - Properly formats and displays task information | 1.0 |
| **Update Task Function** | 2.5 |
| - Accurately identifies the task to update | 1.0 |
| - Correctly updates task details | 1.0 |
| - Includes input validation | 0.5 |
| **Delete Task Function** | 2.5 |
| - Accurately identifies the task to delete | 1.0 |
| - Correctly removes task from the list | 1.0 |
| - Includes input validation | 0.5 |

**2. Implementation (15 Marks)**

* **Python Code for CRUD Functionalities**

| **Criteria** | **Marks** |
| --- | --- |
| **Add Task Function** | 3.75 |
| - Correctly adds a new task to the list | 2.0 |
| - Captures all task details (name, description, priority, due date) | 1.0 |
| - Input validation implemented | 0.75 |
| **View Tasks Function** | 3.75 |
| - Correctly displays all tasks | 2.0 |
| - Proper formatting of output | 1.0 |
| - Handles empty task list gracefully | 0.75 |
| **Update Task Function** | 3.75 |
| - Accurately updates selected task | 2.0 |
| - Updates all relevant fields | 1.0 |
| - Input validation implemented | 0.75 |
| **Delete Task Function** | 3.75 |
| - Accurately deletes selected task | 2.0 |
| - Updates task list accordingly | 1.0 |
| - Handles invalid inputs gracefully | 0.75 |
| **Code Quality (Bonus)** | 1 |
| - Proper variable names and comments | 0.5 |
| - Follows Python coding standards | 0.5 |

**Stage 2: Text File Handling for Task Persistence**

**Total Marks: 20**

**1. Implementation of File Handling (15 Marks)**

* **Load Tasks from File**

| **Criteria** | **Marks** |
| --- | --- |
| - Successfully reads tasks from a text file | 4.0 |
| - Correctly parses and populates the task list | 2.0 |
| - Handles exceptions (e.g., file not found) | 1.5 |

* **Save Tasks to File**

| **Criteria** | **Marks** |
| --- | --- |
| - Successfully writes tasks to a text file | 5.0 |
| - Correct data formatting in the text file | 1.0 |
| - Handles exceptions during file write operations | 1.5 |

**2. Testing (5 Marks)**

* **Test Cases for File Handling Operations**

| **Criteria** | **Marks** |
| --- | --- |
| - At least two test cases for loading tasks | 2.5 |
| - At least two test cases for saving tasks | 2.5 |

**Stage 3: Using Dictionaries and JSON File Handling**

**Total Marks: 25**

**1. Implementation (20 Marks)**

* **Transition to Dictionary-Based Storage**

| **Criteria** | **Marks** |
| --- | --- |
| - Tasks stored as dictionaries with correct keys | 5.0 |
| - Includes all task attributes | 2.0 |
| - Data structure allows easy access to task details | 3.0 |

* **Load Tasks from JSON**

| **Criteria** | **Marks** |
| --- | --- |
| - Successfully reads tasks from a JSON file | 4.0 |
| - Correctly deserializes JSON into dictionaries | 2.0 |
| - Handles exceptions (e.g., JSON decode errors) | 1.5 |

* **Save Tasks to JSON**

| **Criteria** | **Marks** |
| --- | --- |
| - Successfully writes tasks to a JSON file | 4.0 |
| - Correctly serializes dictionaries into JSON | 2.0 |
| - Handles exceptions during file write operations | 1.5 |

**2. Testing (5 Marks)**

* **Test Cases Validating JSON Operations**

| **Criteria** | **Marks** |
| --- | --- |
| - At least two test cases for loading from JSON | 2.5 |
| - At least two test cases for saving to JSON | 2.5 |

**Stage 4: Tkinter GUI for Viewing, Searching, and Sorting Tasks**

**Total Marks: 30**

**1. Class Design and Implementation (15 Marks)**

* **Task Class (5 Marks)**

| **Criteria** | **Marks** |
| --- | --- |
| - \_\_init\_\_ method initializes all properties | 2.0 |
| - Properties include name, description, priority, due date | 1.5 |
| - to\_dict method correctly converts task to dictionary | 1.5 |

* **TaskManager Class (5 Marks)**

| **Criteria** | **Marks** |
| --- | --- |
| - \_\_init\_\_ method initializes task list and loads tasks | 2.0 |
| - load\_tasks\_from\_json method functions correctly | 1.5 |
| - get\_filtered\_tasks and sort\_tasks methods function correctly | 1.5 |

* **TaskManagerGUI Class (5 Marks)**

| **Criteria** | **Marks** |
| --- | --- |
| - \_\_init\_\_ method sets up GUI components | 2.0 |
| - setup\_gui method correctly creates GUI layout | 1.5 |
| - Methods for populating tree, applying filters, and sorting work correctly | 1.5 |

**2. GUI Functionality and Design (10 Marks)**

* **Task Display Table**

| **Criteria** | **Marks** |
| --- | --- |
| - Tasks displayed in Treeview widget | 2.5 |
| - Columns include name, description, priority, due date | 1.0 |
| - Proper formatting and alignment | 1.5 |

* **Search and Filter Functionality**

| **Criteria** | **Marks** |
| --- | --- |
| - Entry fields and dropdowns for filtering present | 1.5 |
| - Filters correctly apply to task list | 1.0 |
| - User input validation | 0.5 |

* **Sorting Functionality**

| **Criteria** | **Marks** |
| --- | --- |
| - Column headers are clickable for sorting | 2.0 |
| - Sorting updates the display correctly | 0.5 |

* **Buttons and Interactivity**

| **Criteria** | **Marks** |
| --- | --- |
| - Buttons for 'Filter' and 'Sort' function correctly | 1.5 |
| - GUI is responsive and user-friendly | 1.0 |

**3. Documentation and User Instructions (5 Marks)**

* **GUI Interface Explanation**

| **Criteria** | **Marks** |
| --- | --- |
| - Clear description of GUI components | 1.5 |
| - Screenshots included (if applicable) | 1.0 |

* **Instructions for Search and Sort Functionalities**

| **Criteria** | **Marks** |
| --- | --- |
| - Step-by-step usage guide | 1.5 |
| - Tips for best user experience | 1.0 |

**4. Code Quality (Up to 2 Bonus Marks)**

* **Code Organization and Readability**

| **Criteria** | **Marks** |
| --- | --- |
| - Proper use of classes and methods | 1.0 |
| - Code is well-commented and follows PEP 8 standards | 1.0 |

**General Guidelines**

* **Timely Submission**
  + Assignments submitted after the deadline may incur penalties according to the course's late submission policy.
* **Originality and Academic Integrity**
  + Ensure that all work submitted is your own and properly cited where necessary.
* **Error Handling**
  + Programs should handle unexpected inputs gracefully without crashing.
* **Testing**
  + Include test cases and evidence of testing where required.
* **Documentation**
  + Provide clear and concise documentation for your code, explaining how each part functions.