Final Project Instructions

Introduction

As we conclude the semester, you will apply all the programming concepts you've learned by developing a comprehensive final project. This project is designed to incorporate **every topic covered in the course**, ensuring you have a strong grasp of Python programming, problem-solving, and GUI development using Tkinter.

Project Overview

You will **choose ONE** of the following three project options to implement:

1. **Personal Finance Management**
2. **Education**
3. **Productivity**

Each project requires you to create a Python application with a Graphical User Interface (GUI) using Tkinter and to apply all the programming concepts covered throughout the semester.

General Requirements for All Projects

Regardless of the project you choose, your application must:

- **Use Tkinter for GUI development**, including multiple windows or frames.
- **Incorporate data types**, variables, expressions, and statements appropriately.
- **Utilize operators**, including arithmetic, comparison, logical, and assignment operators.
- **Import and use built-in modules** and/or **create custom modules** as needed.
- **Define and use functions**, including at least one example of recursion.
- **Manipulate strings**, including slicing and string methods.
- **Use arrays (lists)** and perform operations on them.
- **Implement control flow statements**, including conditionals and loops (for and while).
- **Handle exceptions** using try, except, and finally blocks.
- **Read from and write to files** for data persistence.
- **Use data structures** such as lists, dictionaries, tuples, and sets where appropriate.
- **Include comments** and follow best practices for code readability.
- **Provide thorough documentation** following the specified format.

Specific Project Requirements and Deliverables

1. Personal Finance Management

Project Name: Expense Tracker

Problem Solved: Helps users track and categorize their daily expenses, offering a visual interface to manage their spending habits.

Specific Requirements

1. **GUI Components:**
   - **Main Window:**
     - Displays a menu with options to **Add Expense**, **View Summary**, and **Exit**.
   - **Add Expense Window:**
     - Fields to input **expense amount**, **category** (e.g., Food, Transportation, Entertainment), and **description**.
     - **Drop-down menu** or **radio buttons** for selecting categories.
     - **"Add Expense" button** to submit the data.
   - **View Summary Window:**
     - Displays **total expenses by category** in a table format.
     - Option to view expenses over a **specific date range**.
     - **(Optional for Extra Credit):** Include a **visual representation** (use `matplotlib` or similar library) **(Requires Research)**.
   - **Data Persistence: (Optional for Extra Credit)**
     - **Save expenses to a file** (e.g., .txt file).
     - **Load expenses from the file** when the application starts.
2. **Programming Concepts Application:**
   - **Data Types and Variables:**
     - Use appropriate data types (`float` for amount, `str` for descriptions).
   - **Functions:**
     - Create functions for **adding expenses**, **calculating summaries**, and **generating reports**.
   - **Modules:**
     - **Import built-in modules**
   - **Control Flow and Loops:**
     - Use loops to **iterate over expenses**.
     - Implement conditionals for **data validation**.
   - **Strings:**
     - Manipulate strings for **formatting outputs** and messages.
   - **Arrays (Lists):**
     - Store expenses in a **list** or a list of dictionaries.
   - **Dictionaries:**
     - Use dictionaries to represent individual expenses with keys like `'amount'`, `'category'`, `'date'`, etc.
   - **Tuples and Sets:**
     - Use tuples for fixed data structures if needed.
     - Use sets to find **unique categories**.
   - **Error and Exception Handling:**
     - Handle exceptions during **file operations** and **data conversions**.
   - **File Handling:**
     - Read and write expenses to a **file** for data persistence.
   - **Comments and Code Organization:**
     - Comment your code thoroughly.
     - Organize code into **functions** and possibly **classes**.

3. **Deliverables**

   - **Executable Python Program:**
     - The program should run without errors.
     - All features must be fully implemented and functional.
   - **Source Code:**
     - Fully commented and well-organized code.
     - Include any custom modules in separate files.
   - **Documentation:**
     - Follow the specified formatting and content requirements.
     - Include screenshots of the application in use.
     - Provide explanations for how each part of your code works, referencing specific programming concepts.

---

2. Education

Project Name: Quiz Master

Problem Solved: A simple GUI-based quiz application for students to practice and test their knowledge interactively.

Specific Requirements

1. **GUI Components:**

   - **Main Menu:**
     - Options to **Start Quiz**, **View High Scores**, **Manage Questions**, and **Exit**.
   - **Quiz Window:**
     - Display one **multiple-choice question** at a time.
     - Four options as **radio buttons**.
     - **"Submit Answer" button**.
     - Immediate feedback after each question (correct/incorrect).
     - **Progress indicator** showing question number out of total.
   - **Score Tracking:**
     - Keep track of the user's **score** throughout the quiz.
     - Display **final score** at the end of the quiz.
     - Option to **save the score** with the user's name.
   - **High Scores Window:**
     - Display a list of **high scores** saved from previous sessions.
   - **Question Management (Optional for Extra Credit):**
     - **Manage Questions Window:**
       - Add, edit, or delete questions.
       - Fields to input **question text**, **four options**, and **correct answer**.
     - Save custom question sets to a **file**.
     - Load custom question sets at startup.

2. **Programming Concepts Application:**

   - **Data Types and Variables:**

        - Use appropriate data types for questions, options, and scores.
    - **Functions:**
        - Create functions for **loading questions**, **handling user input**, and **updating scores**.
    - **Modules:**
        - **Import built-in modules** like `random` (for shuffling questions)
        - **Create a custom module** for question management.
    - **Control Flow and Loops:**
        - Use loops to **iterate through questions**.
        - Implement conditionals to **check answers**.
    - **Strings:**
        - Manipulate strings for **displaying questions** and options.
    - **Arrays (Lists):**
        - Store questions in a **list**.
    - **Dictionaries:**
        - Use dictionaries to represent questions with keys like `'question'`, `'options'`, `'correct_answer'`.
    - **Tuples and Sets:**
        - Use tuples for immutable data.
        - Use sets to manage **unique high scores**.
    - **Comments and Code Organization:**
        - Comment your code thoroughly.
        - Organize code into **functions** and possibly **classes**.

3. **Deliverables**

    - **Executable Python Program:**
        - The program should run without errors.
        - All features must be fully implemented and functional.
    - **Source Code:**
        - Fully commented and well-organized code.
        - Include any custom modules in separate files.
    - **Documentation:**
        - Follow the specified formatting and content requirements.
        - Include screenshots of the application in use.
        - Provide explanations for how each part of your code works, referencing specific programming concepts.

---

3. Productivity

Project Name: Task Organizer

Problem Solved: Helps users manage their tasks effectively by organizing and prioritizing their daily activities.

Specific Requirements

1. **GUI Components:**

    - **Main Window:**

        - Displays a list of current tasks.
        - Buttons or menu options to **Add Task**, **Mark as Completed**, **View Completed Tasks**, and **Exit**.
    - **Add Task Window:**
        - Fields to input **task description**, **due date**, and **priority level** (High, Medium, Low).
        - **Drop-down menu** or **radio buttons** for selecting priority.
        - **"Add Task" button** to submit the data.
    - **Task List Display:**
        - Show tasks in an organized manner, sorted by **priority** and/or **due date**.
        - Option to **filter tasks** by priority or due date range.
    - **Completed Tasks Window:**
        - Display a list of tasks marked as **completed**.
        - Option to **restore** or **permanently delete** tasks.
    - **Data Persistence (Optional for Extra Credit):**
        - **Save tasks to a file**.
        - **Load tasks from the file** when the application starts.

2. **Programming Concepts Application:**

    - **Data Types and Variables:**
        - Use appropriate data types for task details.
    - **Operators:**
        - Use comparison operators for **sorting** and **filtering tasks**.
    - **Functions:**
        - Create functions for **adding tasks**, **marking as completed**, and **displaying tasks**.
        - **Include at least one recursive function**, e.g., a function that recursively searches for tasks based on criteria.
    - **Modules:**
        - **Import built-in modules**
        - **Create a custom module** for task management operations.
    - **Control Flow and Loops:**
        - Use loops to **iterate over tasks**.
        - Implement conditionals for **task filtering** and sorting.
    - **Strings:**
        - Manipulate strings for **displaying task details**.
    - **Arrays (Lists):**
        - Store tasks in a **list**.
    - **Dictionaries:**
        - Use dictionaries to represent tasks with keys like `'description'`, `'due_date'`, `'priority'`, `'status'`.
    - **Tuples and Sets:**
        - Use tuples for fixed task attributes.
        - Use sets to manage **unique priorities**.
    - **Comments and Code Organization:**
        - Comment your code thoroughly.
        - Organize code into **functions** and possibly **classes**.

3. **Deliverables**
   - **Executable Python Program:**
     - The program should run without errors.
     - All features must be fully implemented and functional.
   - **Source Code:**
     - Fully commented and well-organized code.
     - Include any custom modules in separate files.
   - **Documentation:**
     - Follow the specified formatting and content requirements.
     - Include screenshots of the application in use.
     - Provide explanations for how each part of your code works, referencing specific programming concepts.

Documentation Requirements (Applies to All Projects)

Prepare a detailed document explaining your project. The documentation must include the following sections and adhere to the specified formatting requirements.

Formatting Requirements:

- **Title Page (Separate Page):**
  - **Project Title**
  - **Your Name**
  - **Date**
  - **Class Period**
- **Purpose:**
  - Explain the purpose of the project in **two sentences**.
  - Clearly state what your program does and what problem it solves.
- **Procedures and Explanation:**
  - Provide a **step-by-step explanation** of how you developed your program.
  - Discuss the **main components** and how they work together.
  - **Explain how you applied each programming concept covered in the course**, providing specific examples from your code:
    - **Data Types and Variables**
    - **Operators**
    - **Functions (including recursion)**
    - **Modules (built-in and custom)**
    - **Control Flow (conditionals and loops)**
    - **Strings and String Manipulation**
    - **Arrays (Lists), Dictionaries, Tuples, Sets**
  - Reference **code snippets** or functions that illustrate the use of these concepts.
- **Pictures of Code for Each Explained Part:**
  - Include **code snippets or screenshots** for each part you explain.
  - Ensure the code is **legible and properly formatted**.

    - **Label each snippet** for clarity.

- **Results and Outputs Including Screenshots:**
    - Provide **screenshots** of your program in action.
    - Show examples of **each feature working** (e.g., adding an expense, answering a quiz question).
    - Explain what **each screenshot** demonstrates.

- **Conclusion:**
    - **What were the results?**
        - Summarize the outcomes of your project.
        - Reflect on whether it met your initial objectives.
    - **What was the error?**
        - Discuss any challenges or errors you encountered.
        - Explain how you resolved them or what issues remain.
    - **How would you improve?**
        - Suggest possible enhancements or alternative approaches.
        - Reflect on what you learned during the project.

Submission Guidelines

- **Deadline:** December 15th @ 11:59 PM EST
- **Format:**
    - Submit your code files (`.py`) and documentation in a **single compressed folder** (`.zip`).
    - Ensure all files are properly named with your **name** and **project title**.

Grading Criteria

Your project will be graded out of **100 points**, divided as follows:

- **Code Implementation (70 points)**
    - **Functionality:** 40 points
        - Full implementation of all specified features.
        - GUI is user-friendly and fully functional.
        - Data persistence works correctly.
        - Input validation and error handling are implemented.
    - **Code Quality and Best Practices:** 20 points
        - Code is well-organized with appropriate functions and/or classes.
        - Appropriate use of programming concepts.
        - Code is efficient with no unnecessary complexity.
    - **Comments and Internal Documentation:** 10 points
        - Code is thoroughly commented.
        - Proper indentation and naming conventions are used.

- **Documentation (30 points)**
    - **Formatting and Structure:** 10 points
        - Title page includes all required elements.

        - Purpose, procedures, and conclusion are clearly presented.
        - Includes code snippets and screenshots as specified.
    - **Detailed Explanation of Code Implementation:** 10 points
        - Thoroughly explains how the code works.
        - Clearly explains which programming concepts are used and how.
    - **Conclusion:** 10 points
        - Results are clearly stated.
        - Errors and challenges are discussed.
        - Suggestions for improvement are thoughtful and insightful.

Note: Detailed rubrics for each section are provided separately. Ensure you review them to understand how each component will be assessed.

Extra Credit Opportunities

You can earn up to **15 extra points** by:

- **Adding additional features** beyond the specifications.
- Enhancing the **GUI** with advanced elements (e.g., animations, custom widgets).
- Implementing **innovative solutions** or optimizations.
- Implementing the Extra Credit portions of the selected project.

Academic Integrity

- **Original Work:** Your submission must be your own work.
- **Plagiarism:** Copying code or documentation from others or from Generative AI models (ChatGPT, Copilot, Gemini, etc.) will not be tolerated.
- **Assistance:** You may seek help for debugging or understanding concepts but do not share code. Identical code or documentation will not be accepted.

Tips for Success

- **Plan Ahead:** Start early to give yourself ample time to work on the project.
- **Understand the Requirements:** Read the project description and requirements carefully.
- **Map Concepts to Requirements:** Make a list of course topics and ensure you incorporate each one into your project.
- **Test Thoroughly:** Test your program with various inputs to ensure it works as expected.
- **Review Your Work:** Proofread your documentation and review your code for any errors.
- **Ask Questions:** If you're unsure about any aspect of the project, don't hesitate to ask for clarification.

Good luck, and I look forward to seeing your comprehensive applications of Python programming!