Assignment Specification: Personal Finance Tracker (Using Only Lists)

Assignment Specification: Personal Finance Tracker (List-Based with JSON Serialization)

Overview

This assignment tasks you with developing a Personal Finance Tracker using Python. Focused on fundamental programming principles like lists, loops, functions, input/output, and input validation, you'll employ list manipulations without the use of dictionaries. Your application will support CRUD operations for managing financial transactions, utilizing JSON for data persistence. This project aims to bolster your understanding of data handling, program design, and testing in a practical context.

Objectives

1. Apply essential Python programming constructs.
2. Manage data using list manipulations.
3. Implement CRUD operations on financial transactions, leveraging JSON serialization for data storage.
4. Conduct thorough testing and validation to ensure program robustness.

Detailed Steps and Marking Scheme

Total Marks: 100

1. **Design (25 Marks)**: Outline your approach using pseudocode, detailing the structure of your list-based data and the flow of the application's features. This design phase is crucial for planning how to implement your program's functionality effectively.
    - **Data Structure Design (10 Marks)**: Present a clear and logical design for how transactions will be stored and manipulated using nested lists.
    - **Pseudocode for Core Features (15 Marks)**: Provide pseudocode for each feature (Add, View, Update, Delete Transactions, Summary, File Operations), demonstrating a thorough understanding of the intended program flow and operations.
2. **Implementation and Documentation (50 Marks)**: Develop the application according to your design, focusing on readability, modular code, and integration of JSON for data handling.
    - **Functionality (30 Marks)**: Implement the core features as specified, with each (Add, View, Update, Delete, Summary, File Operations) evaluated for correctness and efficiency.
    - **Code Quality (20 Marks)**:

        - **Readability and Organization (5 Marks)**
        - **Efficiency and Optimization (5 Marks)**: Efficient use of list operations and program logic.
        - **Modularity and Use of Functions (5 Marks)**
        - **Documentation (5 Marks)**: Clear, comprehensive external documentation on how to use the program, including setup instructions and feature descriptions.
3. **Comments (10 Marks)**: Your code must be well-commented, explaining the purpose and logic behind significant blocks of code and functions. Comments should enhance readability and understanding, offering insights into your development process.
4. **Test (15 Marks)**: Develop a detailed test plan, execute it, and provide a summary of your findings. This plan should cover various scenarios, including edge cases, to ensure the application behaves as expected under different conditions.
    - **Test Plan and Execution (10 Marks)**: A comprehensive test plan detailing tests for each feature, expected outcomes, and actual results. Marks awarded for the thoroughness of the testing and how well it covers the application's functionality.
    - **Test Summary (5 Marks)**: A summary of the test results, including the number of tests passed, failed, and any bugs or issues identified. Reflection on these results and any adjustments made to the program based on testing should be documented.

Notes

- **Data Structure**: Your choice of data structure will significantly impact the ease of implementation and the performance of your program. Careful consideration should be given to how transactions are stored, accessed, and manipulated.
- **CRUD Operations**: The core functionality of your application, providing intuitive and error-resistant methods for these operations, is essential for a good user experience.
- **File Operations**: JSON serialization and deserialization must be implemented correctly to ensure data persistence between program runs.
- **Testing and Validation**: Comprehensive testing is crucial for identifying and resolving potential issues, ensuring your application is reliable and user-friendly.

Submission Guidelines

Submit all program files, including the Python script(s), JSON file(s), documentation, and test plan summary, through your blackboard.
Ensure your submission is well-organized and includes clear instructions for setting up and running your application.

Sample JSON File Format (only for illustrative purpose and you are expected to design one that suites the most)

```
[
  [1000, "Salary", "Income", "2024-02-01"],
  [150, "Groceries", "Expense", "2024-02-03"]
]
```

In this format:

1. The first element is the amount (numeric).
2. The second element is the category (string).
3. The third element is the type ("Income" or "Expense").
4. The fourth element is the date (string, formatted as "YYYY-MM-DD").

Notes

1. Data Structure: It's crucial to carefully plan how transactions will be stored and accessed using nested lists, as this will affect how you implement features like updating and deleting transactions.
2. CRUD Operations: Emphasize creating intuitive and error-proof methods for performing CRUD operations, as these are central to the application's functionality.
3. File Operations: Ensure the program can save and load the transaction list correctly, preserving data between sessions.
4. Testing and Validation: Pay extra attention to testing edge cases, especially in input validation, to ensure the application is robust and user-friendly.

Sample code to start with (only for illustrative purpose)

```
import json

# Global list to store transactions
transactions = []

# File handling functions
def load_transactions():
  pass

def save_transactions():
  pass

# Feature implementations
def add_transaction():
  pass

def view_transactions():
  pass

def update_transaction():
  pass

def delete_transaction():
  pass

def display_summary():
  pass

def main_menu():
  pass

if __name__ == "__main__":
  main_menu()
```