



**Midterm Project Report**

**Advanced Computer Programming**

**Daily Personal Finance Management System**

**Student Name: Gendhis Nuraini Sukmaningtyas Gunandono**

**Student ID: 112021199**

**Teacher: DINH-TRUNG VU**

**2024-04**

# Introduction

## Github

1. **Personal Github Account**: 112021199-Gendhis
2. **Group Github Account**: FullyMed
3. **Group Project Repository**: Amigo
4. **List of submitted file**s:
   * **112021199 Gendhis Nuraini.py**
   * **112021199 Gendhis Nuraini.docx**

## Topic

We're focusing on creating a finance tracker web service. The main goal is is to simplify the process of recording daily expenses and income, as well as setting and monitoring savings targets. By providing a user-friendly interface, this tool will enable users to maintain a clear record of their financial transactions and promote better financial management. For this project, I will implement the functionality to calculate the total balance for expenses and income and also the total amount of money being tracked within the system. My work also involves adding features like expense categorization and timestamps for tracking when income and expenses are added. This allows users to view their finances based on income or expenses and see detailed timestamps for each transaction.

## Project Overview

This Python program is demonstrates basic object-oriented programming concepts and provides a simple way to manage personal finances through command-line interactions, including recording income, categorizing expenses, and calculating financial summaries. The program encapsulates financial data and operations within well-defined classes and functions, enabling efficient and structured financial management.

1. **Data Classes (Transaction, Expense, MoneyData):**

* **Transaction:** Represents a financial transaction with attributes for amount and timestamp.
* **Expense:** Extends Transaction to include a category attribute for expenses.
* **MoneyData:** Manages financial data including lists of income transactions (income) and dictionaries of categorized expense transactions (expenses), along with a maximum expense limit (max\_expense).

1. **Functionality:**

* **Income Management:**

1. add\_income(amount): Adds a new income transaction with the specified amount and timestamp.
2. remove\_income(index): Removes an income transaction by its index.
3. edit\_income(index, new\_amount): Edits the amount of an existing income transaction.
4. view\_income(): Displays all recorded income transactions with their amounts and timestamps.
5. calculate\_avg\_revenue(): Calculates the average income amount per transaction.
6. calculate\_total\_income(): Calculates the total amount of income.

* **Expense Management:**

1. add\_expense(amount, category): Adds a new expense transaction under the specified category, if it does not exceed the maximum expense limit.
2. remove\_expense(category, index): Removes an expense transaction from a specific category by its index.
3. view\_expenses(): Displays categorized expenses with their amounts and timestamps.
4. calculate\_avg\_expenditures(): Calculates the average expenditure amount per category.
5. calculate\_total\_expense(): Calculates the total amount of expenses.

* **Financial Calculations:**

1. calculate\_total\_money(): Calculates the total available money (income minus expenses).
2. **User Interaction (Money Class):**

* The Money class provides a command-line menu (handle\_menu\_choice() and run()) for users to interact with the financial management system.
* Users can choose options to add/remove income, add/remove/edit expenses, view financial summaries, and exit the program.

1. **Execution (if \_\_name\_\_ == "\_\_main\_\_":):**

* Instantiates a Money object to start the financial management system.
* Displays a menu for users to perform various financial operations until they choose to exit.

# Chapter 2 Implementation

## Transaction Class (Transaction)

## Purpose: Represents a financial transaction.

## Attributes:

## amount: The monetary value of the transaction.

## timestamp: The date and time when the transaction occurred.

## Expense Class (Expense)

## Purpose: Extends Transaction to include expense-specific details.

## Additional Attribute:

## category: Represents the category of the expense (e.g., food, transportation, utilities).

## Money Data Class (MoneyData)

## Purpose: Manages financial data including income (list of Transaction objects), expenses (dictionary mapping categories to lists of Expense objects), and max\_expense (float).

## Attribute:

* Income: List of Transaction objects representing income transactions.
* Expenses: Dictionary mapping expense categories to lists of Expense objects representing categorized expenses.
* Max\_expense: Maximum spending limit per month.

## Key Methods and Functions :

* **\_\_init\_\_(self, max\_expense: float):**

Initializes the MoneyData instance with an empty income list, empty expenses dictionary, and a maximum expense limit (max\_expense).

* **add\_income(self, income\_amount: float) -> None:**

Adds a new income transaction to the income list with the specified income\_amount and the current timestamp.

* **remove\_income(self, index: int) -> None:**

Removes an income transaction from the income list based on the provided index.

* **edit\_income(self, index: int, new\_income: float) -> None:**

Edits the amount of an income transaction in the income list at the specified index with new\_income.

* **view\_income(self) -> None:**

Displays all income transactions with their amounts and timestamps.

* **calculate\_avg\_revenue(self) -> float:**

Calculates the average income amount per transaction.

* **calculate\_total\_income(self) -> float:**

Calculates the total income amount.

* **add\_expense(self, expense\_amount: float, category: str) -> None:**

Adds a new expense transaction to the expenses dictionary under the specified category, if the expense amount does not exceed max\_expense.

* **remove\_expense(self, category: str, index: int) -> None:**

Removes an expense transaction from the expenses dictionary based on the specified category and index.

* **view\_expenses(self) -> None:**

Displays all expenses categorized by their categories, including amounts and timestamps.

* **calculate\_avg\_expenditures(self) -> Dict[str, float]:**

Calculates the average expenditure amount per category.

* **calculate\_total\_expense(self) -> float:**

Calculates the total expense amount.

* **calculate\_total\_money(self) -> float:**

Calculates the total available money by subtracting total expenses from total income.

## Money Class (Money)

## Purpose: Provides an interface for user interaction with the finance management system.

## Attribute:

* Main interface for interacting with the finance management system.
* Initializes MoneyData object.
* Implements a menu-driven interface (handle\_menu\_choice()) to execute different actions based on user input.
* Actions include managing income, expenses, and displaying financial statistics.

## Key Methods and Functions :

* **\_\_init\_\_(self):**

Initializes a Money instance by taking the maximum spending limit (max\_expense) per month as input.

* **handle\_menu\_choice(self, choice: str) -> None:**

Executes a specific action based on the user's choice from the menu. This method delegates the actions to appropriate methods in MoneyData.

* **run(self) -> None:**

Runs an infinite loop displaying a menu of options for managing financial data. The loop breaks when the user chooses to finish (choice == "14").

## Main Functions

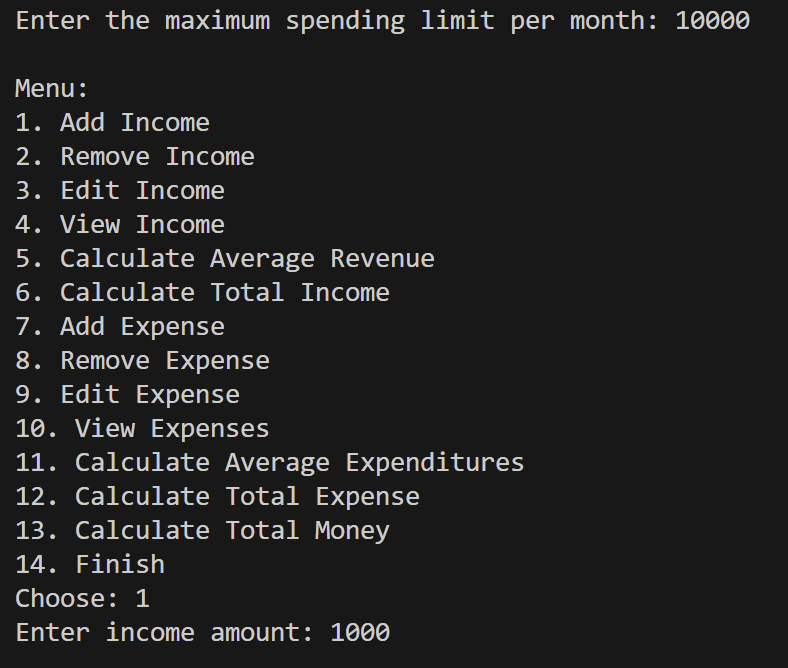
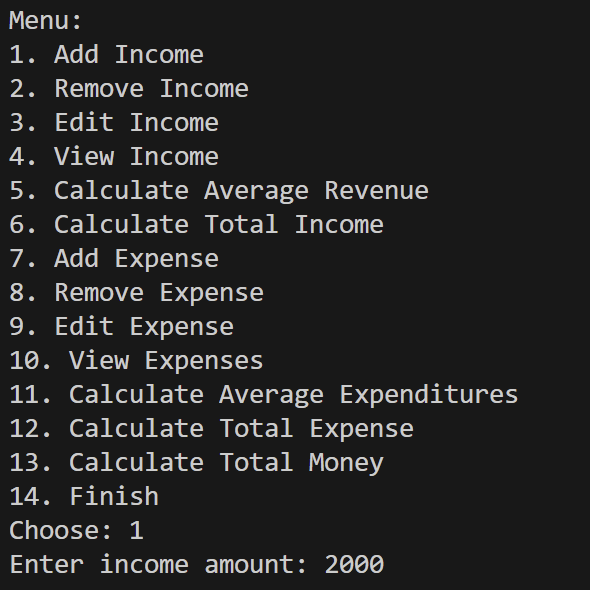
**if \_\_name\_\_ == "\_\_main\_\_"::**

This block creates a Money instance and runs the main loop to interactively manage financial data based on user input through the command-line men

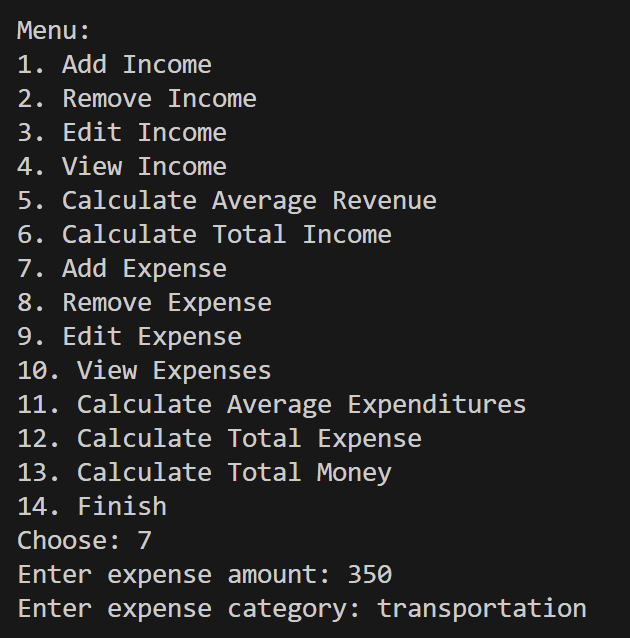
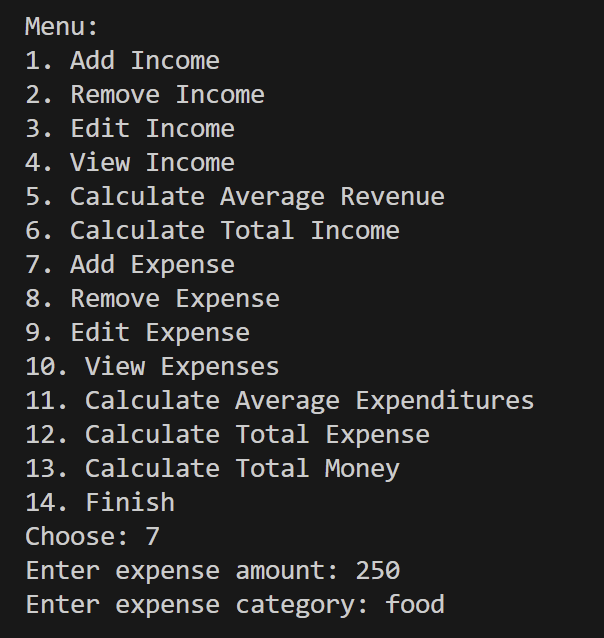
# Chapter 3 Result

## Result

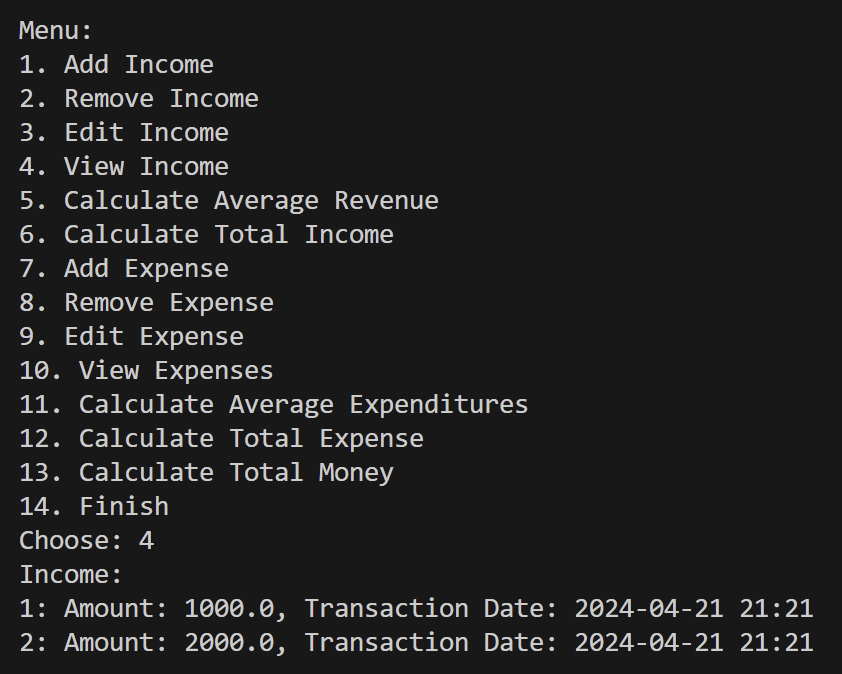
1. In this example, We have successfully added two incomes: 1000 and 2000.

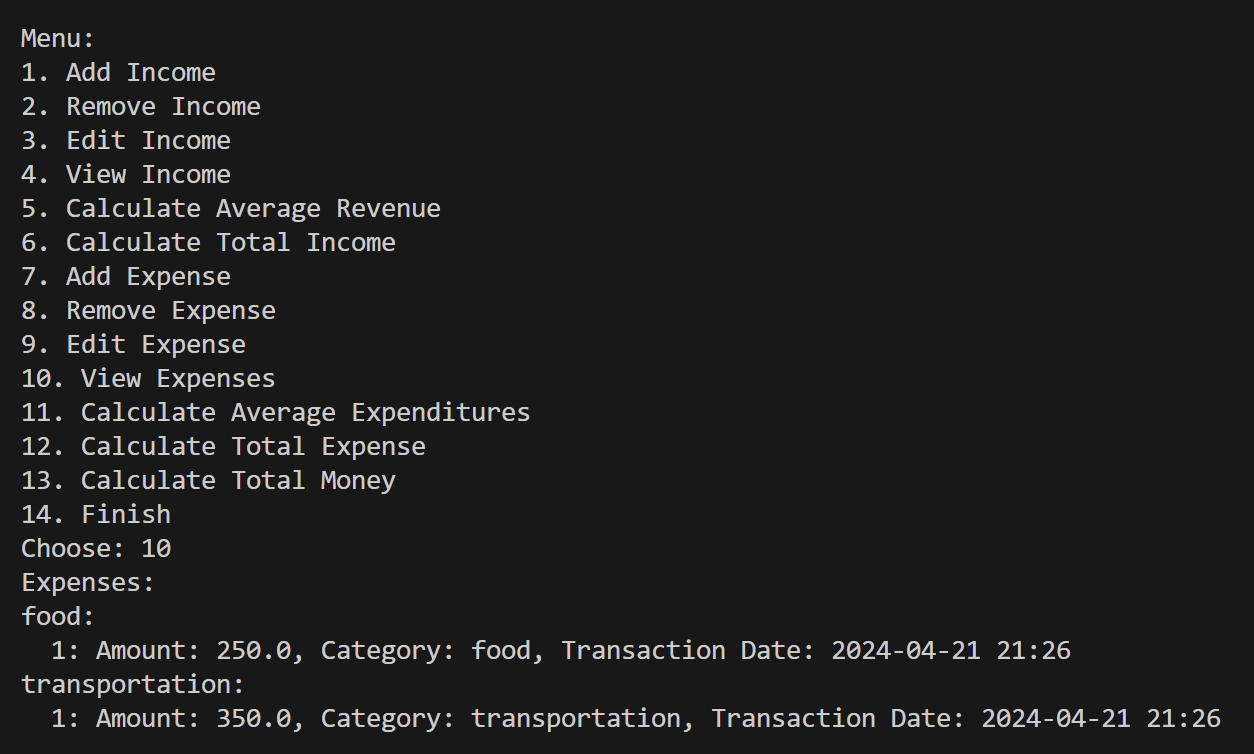


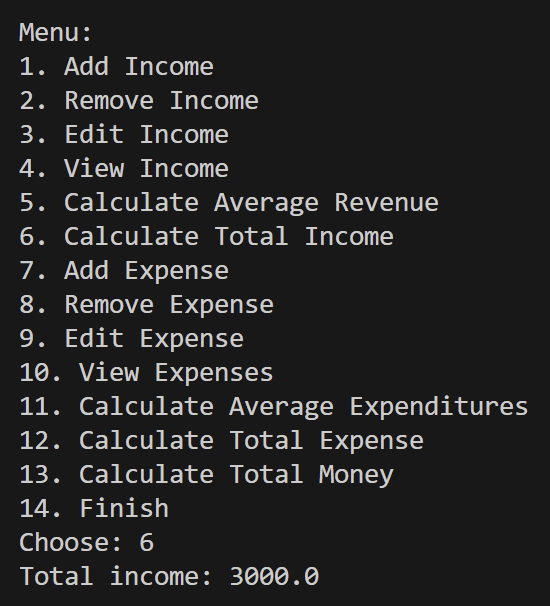
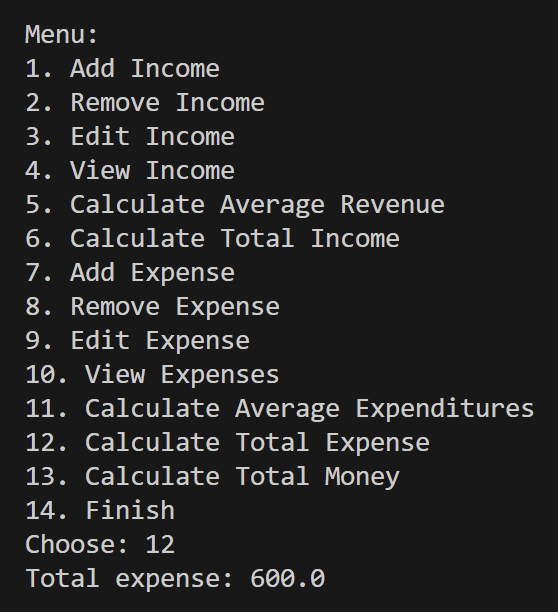
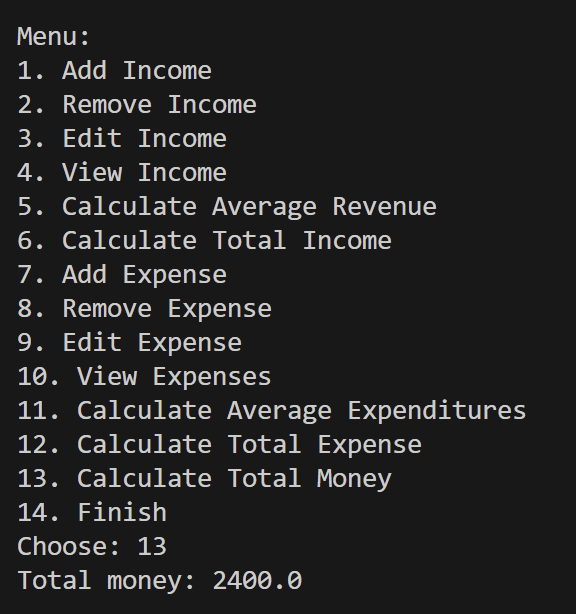
1. In this example, We have successfully added two expenses: the first one is 250 for the food category, and the second one is 350 for the transportation category.



1. The "View Income" and "View Outcome" option displays the current income data, which is stored in the ‘incomes’ and ‘outcome’ lists.

now whenever we add any transaction, the program automatically displays its date, and especially for expenses, it’s even categorized with detailed descriptions. This helps users track their spending and view transaction details easily.



1.  The “Calculate Total Income" option calculates the sum of all current income data and displays it. The "Calculate Total Expense" option also calculates the sum of all current outcome data and displays it.
2. The "Calculate Total Money" option subtracts the total outcome from the total income and displays the result. Since we previously summed the income as 3000 and expenses as 600, the total money is now 2400.

# Chapter 4 Conclusions

## 4.1 Conclusions

The implemented methods and functions in this finance tracker system demonstrate effective utilization of object-oriented programming concepts to manage and analyze personal finances. The system encapsulates financial data within well-defined classes and provides a structured approach to financial management, enabling users to record transactions, categorize expenses, and calculate financial summaries easily. The user interface implemented by the Money class facilitates seamless interaction with the system, empowering users to maintain clear records and make informed financial decisions. Overall, these methods and functions contribute to creating a practical and efficient tool for promoting better financial management practices.