# Minor Project

**Submitted by: Ketha Jagadhish**

**Project: Simple Expense Tracker**

**Objective:**

Create a simple Python project that demonstrates the basic concepts of Python programming, including variables, conditionals, loops, functions, and file handling. The project will allow users to:

1. Add new expenses
2. View the list of expenses
3. Calculate total expenses
4. Save and load expenses from a file

**Project Features:**

1. **Add Expenses:**
    - The user will be able to add an expense with a description and amount.
    - Example: “Lunch” - $12.50
2. **View Expenses:**
    - The user will be able to view all expenses in a list format with serial numbers.
3. **Calculate Total Expenses:**
    - The program will calculate and display the total amount spent.
4. **Save Expenses to a File:**
    - The expenses will be saved in a file (expenses.txt) so that they persist after the program closes.
5. **Load Expenses from a File:**
    - When the program starts, it will load existing expenses from the file.

**Project Outline:**

**Step 1: Define the Main Menu**

- Display a menu with the following options:
    1. Add Expense

2. View Expenses
3. Calculate Total
4. Save and Exit
5. Load Expenses from File

**Step 2: Adding Expenses**

- Create a function add_expense() that allows the user to enter a description and amount for the expense.
- Append the new expense to a list as a tuple (description, amount).

**Step 3: Viewing Expenses**

- Create a function view_expenses() that loops through the list of expenses and prints each expense with a serial number.

**Step 4: Calculating Total Expenses**

- Create a function calculate_total() that sums up the amounts of all expenses and prints the total.

**Step 5: Saving Expenses to a File**

- Create a function save_to_file() that writes the expense data to a text file, one line per expense.

**Step 6: Loading Expenses from a File**

- Create a function load_from_file() that reads the expenses from the file and loads them into the list when the program starts.

**Grading Criteria:**

1. **Functionality**: Does the program correctly add, view, calculate, save, and load expenses?
2. **Code Structure**: Is the code well-structured, modular, and easy to understand?
3. **Correctness**: Does the program handle errors (e.g., invalid input)?
4. **Creativity**: Does the project show creativity, like additional features (e.g., filtering expenses by category)

**Challenges Encountered:**

1. **Handling Invalid Input:**
   - **Challenge**: When adding expenses, users might input invalid data, such as non-numeric values for the amount.
   - **Solution**: We needed to implement error handling using try-except blocks to ensure the user enters a valid number for the amount. If the input is invalid, the program should prompt the user to re-enter a valid value.
2. **Managing Empty Expense List:**
   - **Challenge**: If the user tries to view or calculate expenses without adding any, the program could encounter errors or display nothing.
   - **Solution**: We added checks to verify if the expense list is empty before trying to display or calculate totals. If the list is empty, the program informs the user with an appropriate message.
3. **File Handling (Saving and Loading):**
   - **Challenge**: Saving expenses to a file in a proper format and loading them back can be tricky, especially when handling file I/O operations and ensuring data persistence.
   - **Solution**: We formatted each expense as a CSV line (description,amount) when saving and split the data when loading. We also checked if the file exists before trying to load it to avoid crashes.
4. **Working with Floats for Amounts:**
   - **Challenge**: Working with floating-point numbers (like expense amounts) can lead to precision issues, especially during summation or comparisons.
   - **Solution**: We formatted amounts carefully and ensured that calculations were consistent using float() to avoid precision errors. Proper testing and rounding were added to make sure the amounts displayed accurately.
5. **Maintaining Program Flow:**
   - **Challenge**: Ensuring that the main menu loops properly and only exits when the user chooses to save and exit.
   - **Solution**: We used a while loop for the main menu and checked the user's input to ensure proper navigation. Additionally, we

implemented a clear break condition for the loop when the user chooses to exit.

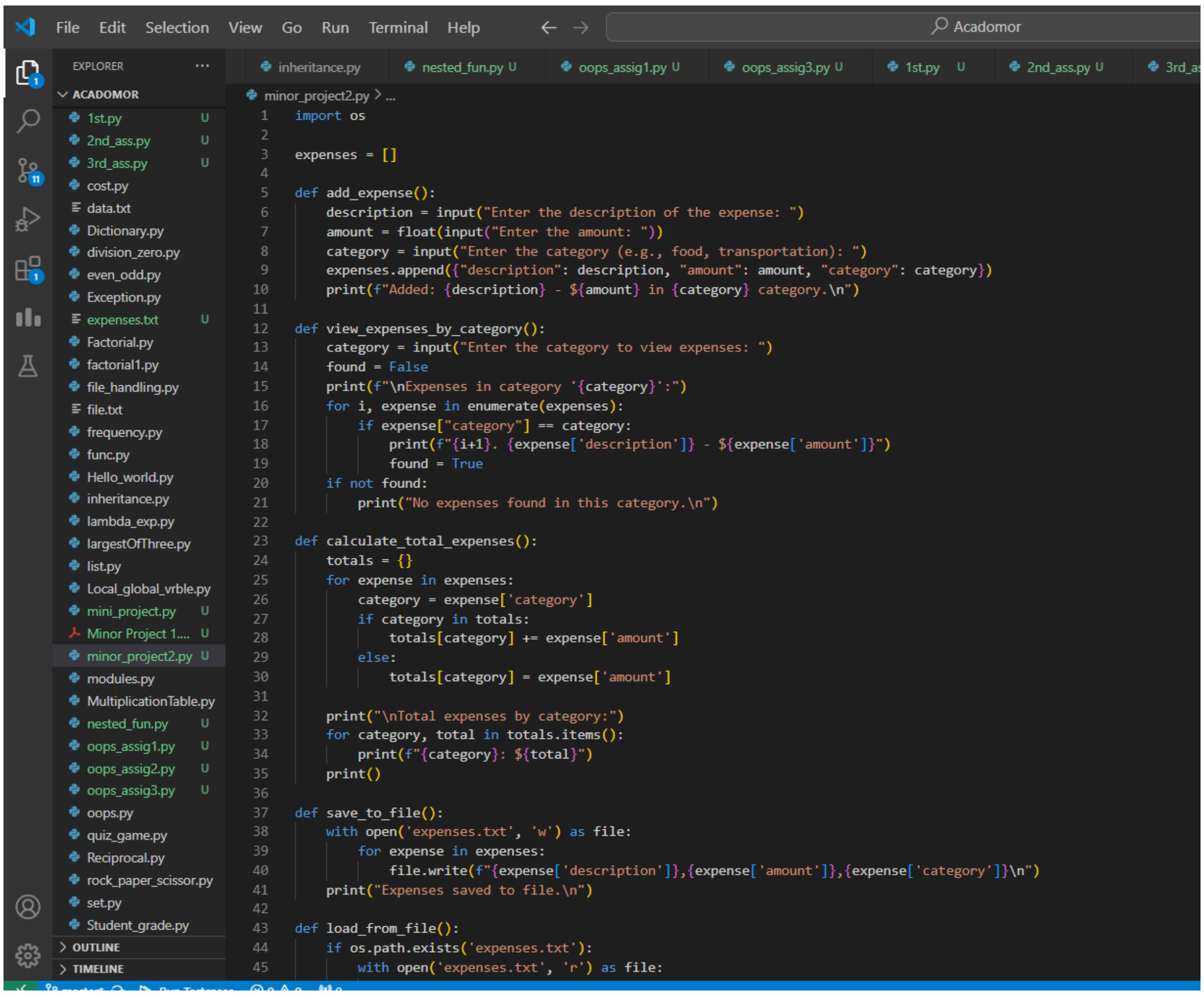
6. **Data Formatting and Display:**
   - **Challenge**: Displaying expenses in a readable format with serial numbers and handling dynamic data could be confusing for users if not done properly.
   - **Solution**: We formatted the output with serial numbers for clarity and ensured that the data displayed is user-friendly and easy to read.

**Conclusion:**

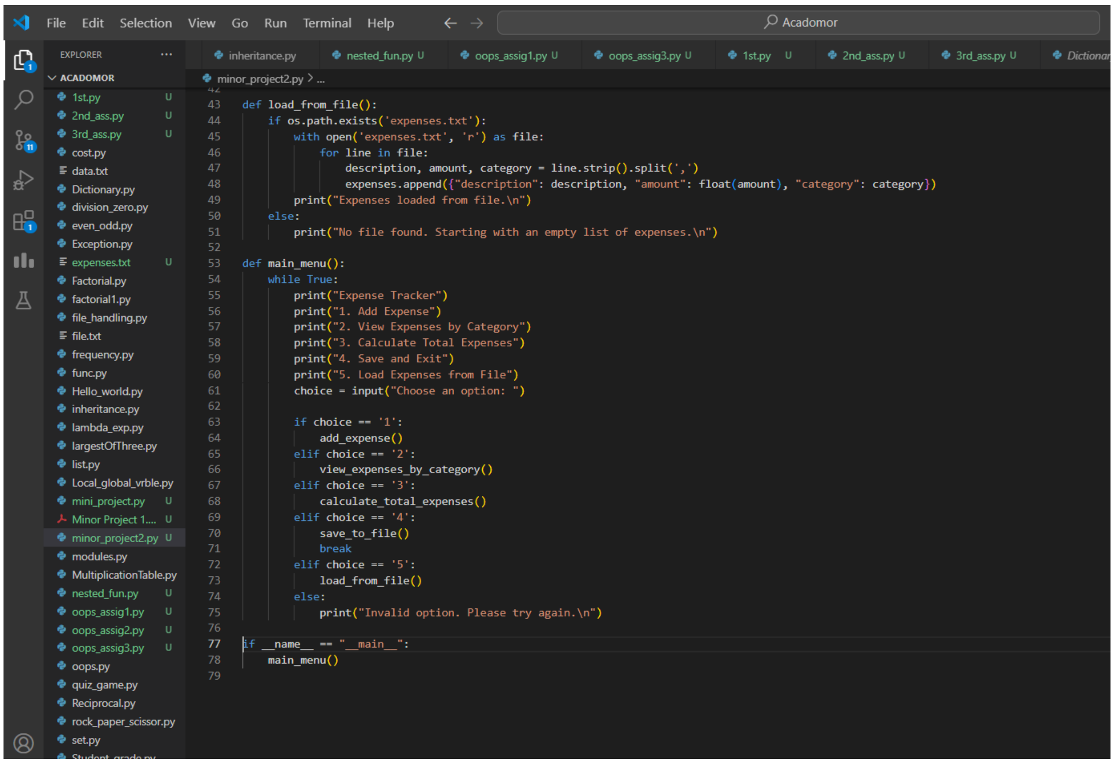
The Expense Tracker project successfully demonstrates the basic concepts of Python programming, including variables, conditionals, loops, functions, and file handling. By completing this project, we have created a simple, yet practical application that allows users to track their daily expenses efficiently. The project covers essential functionalities such as adding, viewing, and calculating expenses, and also ensures data persistence through saving and loading expenses from a file.

This project highlights key programming techniques:

1. **Data Management**: Expenses are stored and manipulated using lists and tuples, demonstrating how to manage dynamic data.
2. **File Handling**: Saving and loading data to and from a file ensures data persistence, allowing users to access their expenses across sessions.
3. **User Interaction**: The menu-driven interface offers a smooth user experience with options to perform various actions.
4. **Error Handling**: The program is designed to manage edge cases, such as invalid inputs and empty lists, improving its robustness.

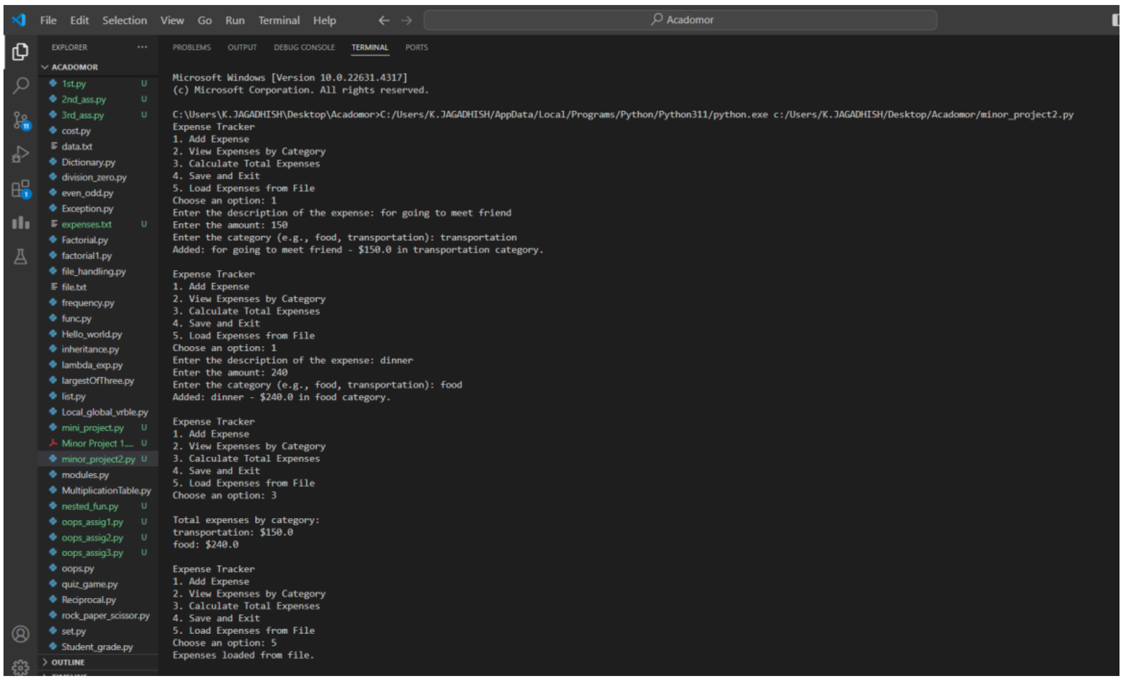


```python
import os

expenses = []

def add_expense():
    description = input("Enter the description of the expense: ")
    amount = float(input("Enter the amount: "))
    category = input("Enter the category (e.g., food, transportation): ")
    expenses.append({"description": description, "amount": amount, "category": category})
    print(f"Added: {description} - ${amount} in {category} category.\n")

def view_expenses_by_category():
    category = input("Enter the category to view expenses: ")
    found = False
    print(f"\nExpenses in category '{category}':")
    for i, expense in enumerate(expenses):
        if expense["category"] == category:
            print(f"{i+1}. {expense['description']} - ${expense['amount']}")
            found = True
    if not found:
        print("No expenses found in this category.\n")

def calculate_total_expenses():
    totals = {}
    for expense in expenses:
        category = expense['category']
        if category in totals:
            totals[category] += expense['amount']
        else:
            totals[category] = expense['amount']

    print("\nTotal expenses by category:")
    for category, total in totals.items():
        print(f"{category}: ${total}")
    print()

def save_to_file():
    with open('expenses.txt', 'w') as file:
        for expense in expenses:
            file.write(f"{expense['description']},{expense['amount']},{expense['category']}\n")
    print("Expenses saved to file.\n")

def load_from_file():
    if os.path.exists('expenses.txt'):
        with open('expenses.txt', 'r') as file:
```



```python
def load_from_file():
    if os.path.exists('expenses.txt'):
        with open('expenses.txt', 'r') as file:
            for line in file:
                description, amount, category = line.strip().split(',')
                expenses.append({"description": description, "amount": float(amount), "category": category})
        print("Expenses loaded from file.\n")
    else:
        print("No file found. Starting with an empty list of expenses.\n")

def main_menu():
    while True:
        print("Expense Tracker")
        print("1. Add Expense")
        print("2. View Expenses by Category")
        print("3. Calculate Total Expenses")
        print("4. Save and Exit")
        print("5. Load Expenses from File")
        choice = input("Choose an option: ")

        if choice == '1':
            add_expense()
        elif choice == '2':
            view_expenses_by_category()
        elif choice == '3':
            calculate_total_expenses()
        elif choice == '4':
            save_to_file()
            break
        elif choice == '5':
            load_from_file()
        else:
            print("Invalid option. Please try again.\n")

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

## Output:



```
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\K.JAGADHISH\Desktop\Acadomor>C:/Users/K.JAGADHISH/AppData/Local/Programs/Python/Python311/python.exe c:/Users/K.JAGADHISH/Desktop/Acadomor/minor_project2.py
Expense Tracker
1. Add Expense
2. View Expenses by Category
3. Calculate Total Expenses
4. Save and Exit
5. Load Expenses from File
Choose an option: 1
Enter the description of the expense: for going to meet friend
Enter the amount: 150
Enter the category (e.g., food, transportation): transportation
Added: for going to meet friend - $150.0 in transportation category.

Expense Tracker
1. Add Expense
2. View Expenses by Category
3. Calculate Total Expenses
4. Save and Exit
5. Load Expenses from File
Choose an option: 1
Enter the description of the expense: dinner
Enter the amount: 240
Enter the category (e.g., food, transportation): food
Added: dinner - $240.0 in food category.

Expense Tracker
1. Add Expense
2. View Expenses by Category
3. Calculate Total Expenses
4. Save and Exit
5. Load Expenses from File
Choose an option: 3

Total expenses by category:
transportation: $150.0
food: $240.0

Expense Tracker
1. Add Expense
2. View Expenses by Category
3. Calculate Total Expenses
4. Save and Exit
5. Load Expenses from File
Choose an option: 5
Expenses loaded from file.
```