**Project:** Expense Tracker

**Project Overview**

The **Expense Tracker** is a Python-based application designed to help users manage their expenses efficiently. This project will involve handling user input, managing and storing data, categorizing expenses, performing data analysis, and implementing a user-friendly interface with error handling. Additionally, proper documentation will be maintained throughout the development process.

**Project Objectives**

* Develop a system that allows users to input their daily expenses.
* Implement a mechanism for storing and managing expense data.
* Categorize expenses for better organization.
* Provide users with insights into their spending patterns, including monthly summaries and category-wise expenditure.
* Create a user-friendly interface for seamless interaction.
* Implement error handling to ensure robustness.
* Document the code effectively for clarity and understanding.

**Requirements and Features**

**User Input**

* Allow users to input their daily expenses, including the amount spent and a brief description.

**Data Storage**

* Use appropriate data structures or file handling techniques to store and retrieve expense data.

**Expense Categories**

* Implement the ability for users to categorize their expenses (e.g., food, transportation, entertainment).

**Data Analysis**

* Provide users with the option to view summaries of their monthly expenses and category-wise expenditure.

**User Interface**

* Create a simple and intuitive user interface to interact with the Expense Tracker.

**Error Handling**

* Include error handling mechanisms to address potential issues during user interaction.

**Documentation**

* Provide clear documentation for your code, explaining the logic behind key functions and overall program structure.

**Day 1: Project Setup and User Input**

* **Goal:** Set up the project structure and implement the basic user input functionality.
* **Tasks:**
  1. Create a new Python file (e.g., expense\_tracker.py).
  2. Define a function to get user input for expenses (amount, description, category).
  3. Start with a simple command-line interface.

Python

def get\_expense\_input():

while True:

try:

amount = float(input("Enter expense amount: "))

if amount <= 0:

print("Amount must be positive.")

continue

break # Exit loop if valid input

except ValueError:

print("Invalid input. Please enter a number.")

description = input("Enter expense description: ")

while True:

category = input("Enter expense category (e.g., food, transport): ").lower()

if not category:

print("Category cannot be empty")

continue

break

return amount, description, category

# Example usage:

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

amount, description, category = get\_expense\_input()

print(f"Amount: {amount}, Description: {description}, Category: {category}")

**Day 2: Data Storage (List of Dictionaries)**

* **Goal:** Implement a way to store the expense data. A list of dictionaries is a good starting point.
* **Tasks:**
  1. Create an empty list to store expenses.
  2. Modify the get\_expense\_input function to add the new expense to the list as a dictionary.

Python

expenses = [] # Initialize an empty list

def get\_expense\_input():

# ... (previous code) ...

expense = {

"amount": amount,

"description": description,

"category": category

}

expenses.append(expense)

return expenses # Return the updated list

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

expenses = get\_expense\_input()

print(expenses) # Print the list of expenses

**Day 3: Basic Data Analysis (Total Expenses)**

* **Goal:** Implement a function to calculate and display the total expenses.
* **Tasks:**
  1. Create a function to calculate the sum of all expense amounts.
  2. Display the total expenses to the user.

Python

def calculate\_total\_expenses(expenses):

total = 0

for expense in expenses:

total += expense["amount"]

return total

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

expenses = get\_expense\_input()

total\_expenses = calculate\_total\_expenses(expenses)

print(f"Total expenses: {total\_expenses}")

**Day 4: Expense Categorization and Analysis**

* **Goal:** Implement category-wise expense tracking and analysis.
* **Tasks:**
  1. Modify the data storage to include categories.
  2. Create a function to calculate expenses per category.

Python

def calculate\_expenses\_by\_category(expenses):

category\_expenses = {}

for expense in expenses:

category = expense["category"]

amount = expense["amount"]

if category in category\_expenses:

category\_expenses[category] += amount

else:

category\_expenses[category] = amount

return category\_expenses

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

expenses = get\_expense\_input() # You can add more expenses for testing

category\_expenses = calculate\_expenses\_by\_category(expenses)

print("Expenses by category:", category\_expenses)

**Day 5: User Interface Enhancement (Menu)**

* **Goal:** Improve the user interface with a menu-driven approach.
* **Tasks:**
  1. Create a main menu loop.
  2. Implement options for adding expenses, viewing total expenses, viewing category-wise expenses, and exiting.

Python

def main\_menu():

while True:

print("\nExpense Tracker Menu:")

print("1. Add Expense")

print("2. View Total Expenses")

print("3. View Category-wise Expenses")

print("4. Exit")

choice = input("Enter your choice: ")

if choice == "1":

expenses.extend(get\_expense\_input()) # extend because get\_expense\_input returns a list

elif choice == "2":

total = calculate\_total\_expenses(expenses)

print(f"Total Expenses: {total}")

elif choice == "3":

category\_expenses = calculate\_expenses\_by\_category(expenses)

print("Category-wise Expenses:", category\_expenses)

elif choice == "4":

break

else:

print("Invalid choice. Please try again.")

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

main\_menu()

**Day 6: Error Handling and Input Validation**

* **Goal:** Improve error handling and input validation.
* **Tasks:**
  1. Add try-except blocks for potential errors (e.g., invalid input).
  2. Validate user input (e.g., positive amounts). You've already started this.

**Day 7: Documentation and Refinement**

* **Goal:** Complete the project with thorough documentation and code cleanup.
* **Tasks:**
  1. Add comments to your code explaining key functionalities.
  2. Write a brief README file explaining how to run the program.
  3. Review and refactor your code for clarity and efficiency.