TRAVEL EXPENSE PLANNER IN PYTHON

Project Report

Submitted by

Divyank Khandelwal

Reg No: 25BCE11112

Department of Computer Science

Vellore Institute of Technology,Bhopal

Abstract

This project is a console-based Travel Expense Planner developed in Python. It helps users track trip expenses across multiple categories such as Accommodation, Transportation, Food and Dining, Activities and Sightseeing, and Miscellaneous. Users can set an overall trip budget, add expenses in different categories, and view a summarized report showing total spending, category-wise distribution, and whether the trip is under or over budget. Data is stored in a JSON file, so information persists across runs. The system uses core Python features such as dictionaries, functions, file handling, and input validation, making it a simple and practical application for managing travel finances.

Contents

List of Tables

Chapter 1

Introduction

Travel involves many expenses that can be difficult to monitor manually. Without a proper system, travelers may overspend, lose track of where their money goes, and find it hard to compare actual spending with the planned budget.

The Travel Expense Planner is a command-line Python application that lets users define a trip budget, record expenses under different categories, add new custom categories, and view an organized summary of spending. The application focuses on simplicity, portability, and persistence using a JSON data file.

Chapter 2

Problem Statement and Objectives

2.1 Problem Statement

Many travelers rely on paper notes or generic mobile apps that may be complex or cluttered. There is a need for a simple tool that:

- Organizes expenses by category.
- Tracks total spending against a predefined budget.
- Persists data between sessions without complex setup.

2.2 Objectives

The main objectives of the project are:

- To design a console-based Python application for tracking travel expenses.
- To enable users to set and update a trip budget.
- To allow adding expenses under predefined and custom categories.
- To generate a category-wise and overall summary of expenses.
- To store and retrieve data using a JSON file for persistence.

Chapter 3

System Requirements

3.1 Hardware Requirements

- Processor: Any modern x86 or ARM processor.
- RAM: Minimum 2 GB.
- Storage: A few MB space for Python and data file.
- Input: Keyboard.
- Output: Console or terminal.

3.2 Software Requirements

- Operating System: Windows, Linux, or macOS.
- Programming Language: Python 3.x.
- Libraries: Standard library modules `time`, `json`, and `os`.
- Editor/IDE: Any Python-capable editor (VS Code, PyCharm, IDLE, etc.).

Chapter 4

System Analysis and Design

4.1 Functional Requirements

1. **Set Trip Budget**: User can set or update the total budget.
2. **Add New Expense**: User selects a category and enters an amount.
3. **Manage Categories**: User can add new custom categories.
4. **View Expense Summary**: Shows totals, category-wise breakdown, and budget status.
5. **Data Persistence**: Loads data from and saves data to `travel_expenses.json`.

4.2 Non-functional Requirements

- Usability: Simple, menu-driven interface.
- Reliability: Handles invalid input and missing or corrupted files gracefully.
- Portability: Runs anywhere Python 3 is available.
- Maintainability: Modular functions and clear global state.

4.3 Data Design

- `EXPENSES`: global dictionary mapping category name (string) to amount (float).
- `BUDGET`: global float storing the total trip budget.
- JSON file `travel_expenses.json` with keys `"expenses"` and `"budget"`.

4.4 System Architecture

The architecture consists of:

- **User Interface Layer**: `main_menu()`.
- **Application Logic Layer**: `add_expense()`, `set_budget()`, `manage_categories()`, `show_summary()`.
- **Persistence Layer**: `load_data()` and `save_data()` for JSON handling.

Chapter 5

Implementation Details

5.1 Global State and Configuration

The application uses:

- `DATA_FILE_PATH` for the JSON file path.
- A default `EXPENSES` dictionary with categories such as Accommodation, Transportation, Food and Dining, Activities and Sightseeing, and Miscellaneous.
- `BUDGET` initialized to zero.

5.2 Data Persistence Functions

5.2.1 `load_data()`

Checks if the JSON file exists, loads expenses and budget if possible, and falls back to default values on error.

5.2.2 `save_data()`

Writes the current `EXPENSES` and `BUDGET` into `travel_expenses.json` using `json.dump()`.

5.3 Input Utility Function

5.3.1 `_get_validated_input()`

A helper function that:

- Prompts the user for input.
- Casts the input to `int` or `float`.

- Enforces optional minimum and maximum values.
- Re-prompts on invalid input or returns `None` if input is unavailable.

5.4 Feature Functions

5.4.1 `set_budget()`

Prompts for a non-negative budget value and updates the global `BUDGET`.

5.4.2 `manage_categories()`

Lets the user add new categories, preventing empty names and duplicates.

5.4.3 `add_expense()`

Shows existing categories, allows the user to choose one, then asks for an amount and updates the `EXPENSES` dictionary.

5.4.4 `show_summary()`

Calculates the total expenses, prints a breakdown by category with percentages, and shows whether the user is under or over the budget.

5.5 Main Application Loop

5.5.1 `main_menu()`

Displays the main menu, reads user choices, calls the appropriate functions, and saves data before exiting.

5.5.2 Program Entry Point

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

5.6 Source Code

```
import time
import json
import os       # Used for checking if the data file exists

# --- Configuration ---
DATA_FILE_PATH = "travel_expenses.json"

# --- Global State ---
# Dictionary to hold expenses. Key: Category Name, Value: Total
   Amount Spent
EXPENSES = {
    "Accommodation": 0.0,
    "Transportation": 0.0,
    "Food & Dining": 0.0,
    "Activities & Sightseeing": 0.0,
    "Miscellaneous": 0.0
}

# Global variable for the total trip budget
BUDGET = 0.0

# --- Data Persistence Functions ---

def load_data():
    """
    Loads expenses and budget from the JSON file if it exists.
    Initializes global EXPENSES and BUDGET.
    """
    global EXPENSES, BUDGET
    if os.path.exists(DATA_FILE_PATH):
        try:
            with open(DATA_FILE_PATH, 'r') as f:
                data = json.load(f)

                # Load EXPENSES, merging with default categories if
                   necessary
                loaded_expenses = data.get('expenses', {})

                # Update EXPENSES with loaded data, preserving
                   default categories if missing
```

```
                # This handles cases where new categories were added
                    in the previous session
                EXPENSES.clear()
                EXPENSES.update(loaded_expenses)

                # Load BUDGET
                BUDGET = data.get('budget', 0.0)

            print(f" Data loaded successfully from {DATA_FILE_PATH}.
                Budget:    {BUDGET:.2f}")
            print(f"Loaded {len(EXPENSES)} expense categories.")

        except (IOError, json.JSONDecodeError) as e:
            print(f" Warning: Could not read or decode existing data
                file. Starting fresh. Error: {e}")
            # Resetting to default state
            EXPENSES.update({
                "Accommodation": 0.0, "Transportation": 0.0, "Food &
                    Dining": 0.0,
                "Activities & Sightseeing": 0.0, "Miscellaneous":
                   0.0
            })
            BUDGET = 0.0
    else:
        print(f"    No existing data file found at {DATA_FILE_PATH}.
            Starting fresh.")

def save_data():
    """
    Saves the current expenses and budget to the JSON file.
    """
    global EXPENSES, BUDGET
    data = {
        'expenses': EXPENSES,
        'budget': BUDGET
    }
    try:
        with open(DATA_FILE_PATH, 'w') as f:
            json.dump(data, f, indent=4)
        print(f" All current data saved successfully to {
           DATA_FILE_PATH}.")
```

```
    except IOError as e:
        print(f" Error: Could not save data to file. Changes will be
            lost. Error: {e}")

# --- Input Utility Function ---

def _get_validated_input(prompt, data_type, min_val=None, max_val=
    None):
    while True:
        try:
            # Handle potential EOFError in restricted environments
            user_input = input(prompt)
        except EOFError:
            print("\nError: Input stream unavailable. Shutting down.
                ")
            return None

        try:
            value = data_type(user_input)

            # Check constraints
            if min_val is not None and value < min_val:
                print(f"Input must be at least {min_val}.")
                continue
            if max_val is not None and value > max_val:
                print(f"Input must not exceed {max_val}.")
                continue

            return value
        except ValueError:
            type_name = "whole number" if data_type is int else "
                decimal number"
            print(f"Invalid input. Please enter a valid {type_name}.
                ")

# --- Feature Handlers ---

def set_budget():
    """
    Allows the user to set or update the total trip budget.
    """
```

```
    global BUDGET
    print("\n--- Set Trip Budget ---")
    new_budget = _get_validated_input("Enter the total trip budget (
        e.g., 50000.00):    ", float, min_val=0.0)

    if new_budget is not None:
        BUDGET = new_budget
        print(f" Trip budget set to    {BUDGET:.2f}.")

def manage_categories():
    """
    Allows the user to add new custom expense categories.
    """
    global EXPENSES
    print("\n--- Manage Categories ---")
    print("1. Add New Category")

    choice = _get_validated_input("Enter your choice (1 to Add, or 0
        to cancel): ", int, min_val=0, max_val=1)

    if choice == 1:
        category_name = input("Enter the name of the new category (e
            .g., 'Souvenirs'): ").strip()

        if not category_name:
            print("Category name cannot be empty.")
            return

        if category_name in EXPENSES:
            print(f"Category '{category_name}' already exists.")
        else:
            EXPENSES[category_name] = 0.0
            print(f" Category '{category_name}' added successfully."
                )
    elif choice == 0:
        print("Category management cancelled.")

def add_expense():
    """
    Prompts the user to add an expense amount to a specific category
        .
```

```
    """
    global EXPENSES
    print("\n--- Add New Expense ---")

    # Display categories dynamically (includes newly added ones)
    categories = list(EXPENSES.keys())
    print("Available Categories:")
    for i, category in enumerate(categories):
        print(f"  {i+1}. {category}")

    # 1. Get Category Choice
    max_choice = len(categories)

    choice = _get_validated_input(
        f"Enter the number of the category (1-{max_choice}, or 0 to
            cancel): ",
        int,
        min_val=0,
        max_val=max_choice
    )

    if choice is None: return
    if choice == 0:
        print("Expense addition cancelled.")
        return

    category_name = categories[choice - 1]

    # 2. Get Amount
    amount = _get_validated_input(
        f"Enter the amount for {category_name} (e.g., 55.50):    ",
        float,
        min_val=0.0
    )

    if amount is None: return

    # Update the expense dictionary
    EXPENSES[category_name] += amount
    print(f"\n Added    {amount:.2f} to {category_name}.")
```

```
# --- Reporting Function ---

def show_summary():
    """
    Calculates and displays the total expenses, breakdown by
       category, and budget status.
    """
    global EXPENSES, BUDGET
    print("\n==================================")
    print("      TRAVEL EXPENSE SUMMARY      ")
    print("==================================")

    total_expense = sum(EXPENSES.values())

    if total_expense == 0:
        print("No expenses recorded yet.")
        if BUDGET > 0:
            print(f"Trip Budget Set:    {BUDGET:10.2f}")
        print("==================================\n")
        return

    # Display breakdown
    print("\n--- Breakdown by Category ---")
    for category, amount in EXPENSES.items():
        if amount > 0:
            percentage = (amount / total_expense) * 100
            print(f"{category:<25} :    {amount:10.2f} ({percentage
               :5.1f}%)")

    # Display total
    print("-" * 34)
    print(f"TOTAL SPENT EXPENSES:    {total_expense:10.2f}")

    # Budget Tracking Section
    if BUDGET > 0:
        remaining = BUDGET - total_expense
        print("\n--- Budget Tracking ---")
        print(f"Trip Budget Set:       {BUDGET:10.2f}")

        if remaining >= 0:
```

```
            print(f"Amount Remaining:      {remaining:10.2f} (Under
                Budget)")
        else:
            # Show the absolute value of the amount over budget
            print(f"Amount OVER Budget:    {abs(remaining):10.2f} (
                Over Budget)")

    print("==================================\n")

# --- Main Application Loop ---

def main_menu():
    """
    Main loop for the expense planner application.
    """
    print("Welcome to the Simple Travel Expense Planner!")
    load_data() # Load data when the application starts

    while True:
        time.sleep(0.5)
        print("\n--- Main Menu ---")
        print("1. Add New Expense")
        print("2. View Expense Summary")
        print("3. Set Trip Budget")
        print("4. Manage Categories")
        print("5. Exit Planner")

        choice = _get_validated_input("Enter your choice (1-5): ",
            int, min_val=1, max_val=5)

        if choice is None:
            choice = 5 # Treat input failure as exit

        if choice == 1:
            add_expense()
        elif choice == 2:
            show_summary()
        elif choice == 3:
            set_budget()
        elif choice == 4:
            manage_categories()
```

```
        elif choice == 5:
            save_data() # Save data when the user chooses to exit
            print("\nThank you for using the planner!")
            break

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

Chapter 6

Testing

6.1 Sample Test Cases

Test Case	Input / Action	Expected Result
First Run	No JSON file present	Program starts with default categories and zero budget.
Set Budget	Set budget to 50000	Summary shows budget 50000 and correct remaining amount.
Add Expense	Add 2000 to Accommodation	Category total and overall total are updated correctly.
Over Budget	Budget 1000, expenses 1500	Summary shows overspending and amount over budget.
Invalid Input	Non-numeric where number required	Program shows error and re-prompts.

Table 6.1: Sample Test Cases

Chapter 7

Advantages, Limitations, and Future Enhancements

7.1 Advantages

- Simple menu-based interface.
- Uses only Python standard library.
- Automatically persists data across runs.
- Supports custom expense categories.

7.2 Limitations

- Only one trip is tracked at a time.
- No GUI or charts.
- No export to CSV or PDF reports.

7.3 Future Enhancements

- Support multiple trips with separate budgets.
- Add a graphical or web-based interface.
- Export summaries to CSV or PDF.
- Visualize spending using charts.

Chapter 8

Conclusion

The Travel Expense Planner in Python demonstrates how a simple console application can help users monitor their travel spending and stay within budget. By combining basic data structures, file handling, and modular design, the project provides a clear view of category-wise spending and overall budget status and can be extended into a more advanced budgeting tool.