

EXPENSE TRACKER REPORT

PYTHON PROJECT

VITyarthi PROJECT

NAME : KAASHISH ADLAK

REG NO.: 25MIB10007

1. Introduction

Managing day-to-day expenses is an essential task for individuals and businesses alike. Most people struggle to track their spending habits due to manual methods such as notebooks or basic spreadsheets. To overcome these challenges, a computer-based solution offers a more efficient, organized, and error-free approach.

This project presents a Python-based Expense Tracker Application built using Tkinter for the graphical user interface, SQLite as the database for storing expense records, and tkcalendar to facilitate date selection. The application enables users to add, update, view, and delete expenses in a user-friendly manner while maintaining a persistent database.

2. Problem Statement

Many individuals lack a systematic way to track their daily expenditures. Manual tracking methods are time-consuming, prone to errors, and often lack proper organization and retrieval features. Without proper tracking, users cannot analyze their spending patterns or maintain financial discipline.

The main problem addressed by this project is the absence of an easy-to-use, computerized system that allows users to record, edit, and manage expenses efficiently, ensuring accuracy and quick access to financial records.

3. Functional Requirements

Functional requirements describe what the system should do.

3.1 Add Expense

The user must be able to enter details such as date, payee, description, amount, and mode of payment.

The application should store this data in an SQLite database.

3.2 View Expense

The system must display all stored expenses in a tabular Tree View.

The user must be able to select an expense to view its detailed information.

3.3 Edit Expense

The user must be able to update previously stored expense information.

Updated records must be saved in the database.

3.4 Delete Expense

The user must be able to delete a selected expense.

The system must also offer an option to delete all expenses at once.

3.5 Clear Fields

The input form should reset to default values when the "Clear Fields" action is triggered.

3.6 Convert Expense to Words

The system allows users to convert a selected expense into a readable sentence.

This helps users understand the record linguistically.

3.7 Date Selection

The calendar widget must allow easy date picking to avoid format errors.

4. Non-Functional Requirements

Non-functional requirements define the system's quality attributes.

4.1 Usability

The interface should be simple and intuitive.

Font style, button size, and layout should support easy navigation.

4.2 Performance

The system should quickly load, retrieve, and update expense data.

Operations like database read/write must be fast.

4.3 Reliability

Data stored in SQLite must be persistent.

The system should not crash during common operations such as adding or editing entries.

4.4 Maintainability

The source code should be modular, with separate functions handling specific tasks.

Clear naming conventions and comments should help future developers modify the system.

4.5 Portability

As the system uses Python and SQLite, it can run on multiple operating systems (Windows, macOS, Linux).

4.6 Security

Data integrity must be preserved by preventing invalid or incomplete fields when saving.

The system prompts users before deleting records to avoid accidental loss.

5. System Architecture

The system follows a simple three-layer architecture:

5.1 Presentation Layer (User Interface Layer)

Built using Tkinter.

Includes widgets such as Labels, Entry fields, Buttons, OptionMenu, DateEntry, and TreeView.

Supports user interaction for adding, viewing, editing, and deleting expenses.

5.2 Application Logic Layer

Consists of Python functions handling operations:

`add_another_expense()` – Adds new expenses

`edit_expense()` – Updates selected records

`remove_expense()` – Deletes a record

`remove_all_expenses()` – Clears all data

`list_all_expenses()` – Loads all data from database

`view_expense_details()` – Displays details of selected entry

Ensures validation, message prompts, and linking UI with backend.

5.3 Data Layer (Database Layer)

Uses SQLite database named Expense Tracker.db.

Table: ExpenseTracker

Columns: ID, Date, Payee, Description,
Amount, ModeOfPayment

Stores all expenses securely and persistently.

DESIGN DIAGRAMs

Use Case Diagrams



Work Flow Diagram

```

+-----+ | ExpenseTracker | +-----+ | - ID:
Integer | | - Date: Date | | - Payee: String | | - Description: String | | - Amount:
Float | | - ModeOfPayment: String | +-----+ | + addExpense() | | +
deleteExpense() | | + updateExpense() | | + fetchAll() | | + fetchOne() |
+-----+ +-----+ | Database |
+-----+ | + connect() | | + executeQuery() | | + commit() |
+-----+ +-----+ | GUI |
+-----+ | + renderForm() | | + renderTable() | | + clearFields() | |
+ showMessage() | +-----+

```

DFD Level 0

```

+-----+ | USER | +-----+ | v
+-----+ | Expense Tracker | +-----+ | v
+-----+ | Database (SQLite) | +-----+

```

DFD Level 1

```

+-----+ | USER | +-----+ | 
+-----+ +-----+ +-----+ | Add Exp | | Edit Exp | | Delete Exp | |
View Expense | +-----+ +-----+ +-----+ +-----+ +-----+ | | | | |
----- Write/Read ----- | v +-----+ | | | |
SQLite Database | +-----+

```

ERD Diagram

```

+-----+ | ExpenseTracker | 
+-----+ | ID (PK) INTEGER | | Date DATETIME |
| Payee TEXT | | Description TEXT | | Amount FLOAT | | ModeOfPayment TEXT |
+-----+

```

GUI Layout Diagram

```

----- | EXPENSE TRACKER (Title Bar) | -----
| TABLE PANEL | | ----- | DATA ENTRY PANEL
| [DateEntry] | +-----+ | | Payee: [Textbox] | | ID | Date | Payee
| ... | | Description: [Textbox] | |-----| | | Amount:
| [Textbox] | | Data Rows... | | Mode: [Dropdown] | +-----+ | |
| [Add Button] | | | [Convert to Words] | |
----- | BUTTONS PANEL (Delete, Edit, Clear, etc.) | -----

```


1. Design Decisions

The following design choices were made to ensure that the Expense Tracker is simple, efficient, and user-friendly:

1.1 Choice of Programming Language

Python was chosen because:

It offers strong GUI support through Tkinter.

It integrates easily with SQLite for backend storage.

It is simple to learn and widely used in software development.

1.2 GUI Framework (Tkinter)

Tkinter was selected as the GUI framework because:

It comes built-in with Python (no additional installation needed).

It supports widgets like buttons, forms, tables, dropdowns, and message boxes.

It is ideal for desktop applications with moderate complexity.

1.3 Database (SQLite)

SQLite was preferred due to:

Lightweight storage in a single .db file.

No server installation required.

Fast read/write operations for small applications.

1.4 TkCalendar Widget

The DateEntry widget from tkcalendar was used to:

Avoid incorrect date formatting.

Provide a calendar-style selection.

1.5 Table Display (TreeView)

TreeView from Tkinter's ttk module was used for:

Structured representation of expenses.

Horizontal and vertical scrolling.

Easy selection of specific records.

1.6 Modular Function Design

Functions like add_another_expense(), edit_expense(), and remove_expense() were separated to:

Improve readability.

Make the code maintainable.

Allow reuse and easy debugging.

2. Implementation Details

Below is a summary of how the system is built internally:

2.1 Database Initialization

A table named ExpenseTracker is created with fields:

ID (Primary Key)

Date

Payee

Description

Amount

Mode of Payment

This ensures all expenses are stored permanently.

2.2 Main Functional Modules

- Add Expense

Takes form input values.

Validates empty fields.

Inserts values into database.

- View Expense

Loads all values from database into TreeView.

Allows selection to view details.

- Edit Expense

Retrieves selected record.

Updates fields in the form.

Saves modified data back to the database.

- Delete Expense

Deletes the selected item or all items.

Confirmation message ensures safety.

- Clear Fields

Resets the input form to default values.

- Convert Expense to Words

Converts numeric and textual details into a readable sentence.

2.3 GUI Layout

The window is divided into:

Data Entry Frame (left side)

Buttons Frame (top center)

TreeView Frame (right side listing the expenses)

2.4 Application Flow

1. User inputs expense details.
2. Data gets stored in the SQLite database.
3. Data is fetched and displayed in the TreeView.
4. User can edit, delete, or view details of each entry.

RESULT

The screenshot shows the Visual Studio Code interface with the file `vityarthifinal.py` open. The code initializes a SQLite database and defines functions for listing all expenses and viewing details of a selected expense. A terminal window is visible at the bottom, showing command-line interactions.

```
File Edit Selection View Go Run Terminal Help ← → Q csevit
RUN AND DEBUG ...
RUN
Run and Debug
To customize Run and Debug create a launch.json file.
Show automatic Python configurations
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ ... | x
POWER SHELL
POWER SHELL
Python
POWER SHELL
POWER SHELL
POWER SHELL
Ln 277, Col 14 Spaces: 4 UFT-8 CRLF {} Python 3.14.0 02:02 23-11-2025
C:\Users\KASHISH ADLAK\OneDrive\csevit> & "C:/Users/KASHISH ADLAK/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/KASHISH ADLAK/OneDrive/csevit/vityarthifinal.py"
c:/Users/KASHISH ADLAK/OneDrive\csevit\vityarthifinal.py:102: DeprecationWarning: The default date adapter is deprecated as of Python 3.12; see the sqlite3 documentation for suggested replacement recipes
connector.execute(
PS C:\Users\KASHISH ADLAK\OneDrive\csevit> & "C:/Users/KASHISH ADLAK/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/KASHISH ADLAK/OneDrive/csevit/vityarthifinal.py"
PS C:\Users\KASHISH ADLAK\OneDrive\csevit>

```

The screenshot shows the Visual Studio Code interface with the same file `vityarthifinal.py` open. The code now includes Tkinter-based user interface logic for displaying expense details and clearing fields. The terminal window continues to show command-line interactions.

```
File Edit Selection View Go Run Terminal Help ← → Q csevit
RUN AND DEBUG ...
RUN
Run and Debug
To customize Run and Debug create a launch.json file.
Show automatic Python configurations
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ ... | x
POWER SHELL
POWER SHELL
Python
POWER SHELL
POWER SHELL
POWER SHELL
Ln 277, Col 14 Spaces: 4 UFT-8 CRLF {} Python 3.14.0 02:03 23-11-2025
C:\Users\KASHISH ADLAK\OneDrive\csevit> & "C:/Users/KASHISH ADLAK/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/KASHISH ADLAK/OneDrive/csevit/vityarthifinal.py"
c:/Users/KASHISH ADLAK/OneDrive\csevit\vityarthifinal.py:102: DeprecationWarning: The default date adapter is deprecated as of Python 3.12; see the sqlite3 documentation for suggested replacement recipes
connector.execute(
PS C:\Users\KASHISH ADLAK\OneDrive\csevit> & "C:/Users/KASHISH ADLAK/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/KASHISH ADLAK/OneDrive/csevit/vityarthifinal.py"
PS C:\Users\KASHISH ADLAK\OneDrive\csevit>
```

The screenshot shows a code editor window with the following details:

- File Path:** vityarthifinal.py
- Code Content:** Python code for expense tracking. It includes functions for clearing fields, removing expenses, and adding another expense. It uses global variables like date, payee, desc, amnt, and MOP.
- Terminal Output:** Shows command-line interactions with Python and SQLite databases.
- Status Bar:** Displays file path, line number (Ln 277), column number (Col 14), spaces (Spaces: 4), encoding (UTF-8), and Python version (3.14.0).

The screenshot shows a code editor window with the following details:

- File Path:** vityarthifinal.py
- Code Content:** Python code for expense tracking, similar to the first one but with different logic for removing all expenses and adding another expense.
- Terminal Output:** Shows command-line interactions with Python and SQLite databases.
- Status Bar:** Displays file path, line number (Ln 277), column number (Col 14), spaces (Spaces: 4), encoding (UTF-8), and Python version (3.14.0).

The screenshot shows a code editor window with the following details:

- File Path:** vityarthifinal.py
- Code Content:** Python code for expense tracking. It includes functions for adding expenses, editing existing expenses, and displaying expense details.
- Toolbars:** RUN AND DEBUG, RUN, Run and Debug, To customize Run and Debug create a launch.json file, Show automatic Python configurations.
- Bottom Status Bar:** Ln 277, Col 14 | Spaces: 4 | UTF-8 | CRLF | {} Python | 3.14.0 | 02:03 | 23-11-2025

The screenshot shows a code editor window with the following details:

- File Path:** vityarthifinal.py
- Code Content:** Python code for expense tracking, similar to the first screenshot but with different logic for handling expense selection.
- Toolbars:** RUN AND DEBUG, RUN, Run and Debug, To customize Run and Debug create a launch.json file, Show automatic Python configurations.
- Bottom Status Bar:** Ln 142, Col 1 | Spaces: 4 | UTF-8 | CRLF | {} Python | 3.14.0 | 02:04 | 23-11-2025

The screenshot shows a Python IDE interface with the following details:

- File**, **Edit**, **Selection**, **View**, **Go**, **Run**, **Terminal**, **Help** menu bar.
- RUN AND DEBUG** button in the top-left corner.
- RUN** dropdown menu open, showing **Run and Debug** and **To customize Run and Debug create a launch.json file.**
- Show automatic Python configurations** button in a blue box.
- Code Editor**: The main area displays Python code for a Tkinter application named `vityarthifinal.py`. The code initializes a root window, sets its title and geometry, creates labels for expense tracking, defines variables for description, amount, payee, and mode of payment, creates frames for data entry and buttons, and places a tree frame. It also handles the placement of elements within the data entry frame.
- Terminal**: Shows command-line output for running the script in PowerShell and Python 3.14, including a deprecation warning about the default date adapter.
- PROBLEMS**, **OUTPUT**, **DEBUG CONSOLE**, **TERMINAL**, **PORTS** tabs at the bottom.
- BREAKPOINTS** section with a list of breakpoints: **Raised Exceptions**, **Uncought Exceptions**, and **User Uncaught Exceptions**.
- System Tray**: Shows battery level (14%), clear icon, and system status icons.
- Bottom Bar**: Includes a search bar, file navigation icons, and system status indicators like ENG US, WiFi, and battery level.

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a file tree with a 'RUN AND DEBUG' folder and a 'RUN' section containing 'Run and Debug' and 'Show automatic Python configurations'.
- Code Editor:** Displays Python code for a file named `vityarthifinal.py`. The code defines a window with several buttons and a treeview frame. It uses Tkinter for the GUI and SQLite for database operations.
- Terminal:** Shows the command-line output of running the script:

```
PS C:\Users\KASHISH ADLAK\OneDrive\csevit> & "C:\Users\KASHISH ADLAK\AppData\Local\Programs\Python\Python314\python.exe" "c:/Users\KASHISH ADLAK\OneDrive\csevit\vityarthifinal.py"
c:/Users\KASHISH ADLAK\OneDrive\csevit\vityarthifinal.py:102: DeprecationWarning: The default date adapter is deprecated as of Python 3.12; see the sqlite3 documentation for suggested replacement recipes
    connector.execute()
PS C:\Users\KASHISH ADLAK\OneDrive\csevit> & "C:\Users\KASHISH ADLAK\AppData\Local\Programs\Python\Python314\python.exe" "c:/Users\KASHISH ADLAK\OneDrive\csevit\vityarthifinal.py"
PS C:\Users\KASHISH ADLAK\OneDrive\csevit>
```
- Bottom Status Bar:** Shows file path (Ln 142, Col 1), spaces, tabs, and encoding (UTF-8). It also includes icons for file operations like save, undo, and redo, and a Python icon.
- Right Sidebar:** Shows a preview of the Python code and a sidebar with icons for powershell, Python, and other tools.

The screenshot shows a Windows desktop environment. In the foreground, a Snipping Tool window is open, displaying a screenshot of a Microsoft Visual Studio Code (VS Code) instance. The VS Code interface includes a sidebar with icons for file operations, a 'RUN AND DEBUG' section, and a 'BREAKPOINTS' section. The main area shows a Python script named 'vityarthifinal.py' with code for creating a GUI window using Tkinter and ttk. The terminal tab at the bottom shows command-line output from a PowerShell session (PS) running on Windows, indicating the script was executed successfully. The taskbar at the bottom of the screen shows various pinned and running applications.

```
263     table.column('#0', width=0, stretch=NO)
264     table.column('#1', width=50, stretch=NO)
265     table.column('#2', width=95, stretch=NO)
266     table.column('#3', width=150, stretch=NO)
267     table.column('#4', width=325, stretch=NO)
268     table.column('#5', width=135, stretch=NO)
269     table.column('#6', width=125, stretch=NO)
270
271     table.place(relx=0, y=0, relheight=1, relwidth=1)
272
273     list_all_expenses()
274
275     # Finalization of the GUI window takes place here. The main loop commences afterward.
276
277     root.update()
278     root.mainloop()
279
280
281
282
283
284
285
286
```

```
PS C:\Users\KASHISH ADLAK\OneDrive\csevit & "C:/Users/KASHISH ADLAK/AppData/Local/Programs/Python/Python314/python.exe" "csevit\vityarthifinal.py"
c:\Users\KASHISH ADLAK\OneDrive\csevit\vityarthifinal.py:102: DeprecationWarning: The default date adapter is deprecated
documentation for suggested replacement recipes
    connector.execute()
PS C:\Users\KASHISH ADLAK\OneDrive\csevit & "C:/Users/KASHISH ADLAK/AppData/Local/Programs/Python/Python314/python.exe" "csevit\vityarthifinal.py"
PS C:\Users\KASHISH ADLAK\OneDrive\csevit
```

OUTPUT

https://youtu.be/dzEkYdu8B_c?si=2iyzs4YyTzuVHITA

3. Testing Approach

Several testing techniques were followed:

3.1 Manual Testing

GUI applications require manual interaction.

The following operations were tested:

Adding new expenses.

Editing existing entries.

Deleting expenses one by one and all at once.

Viewing selected record details.

Ensuring scrollbars work properly.

Checking date selection from the calendar.

3.2 Functional Testing

Each function was tested separately:

Input validation

Correct insertion into DB

Accurate fetching of data

Update and delete operations

3.3 Boundary Testing

Empty fields

Very large entries in description

Zero or negative amount values

Invalid mode of payment selection

3.4 Database Testing

Verified:

Data integrity

Auto-incrementing ID

Correct deletion and refresh in the TreeView

4. Challenges Faced

4.1 Handling Tkinter Layout

Managing multiple frames and widget positions required careful alignment to avoid overlapping.

4.2 Date Formatting Issues

Dates needed conversion between:

TkCalendar format

SQLite storage format

Python datetime object

4.3 Updating TreeView

Refreshing the list required:

Clearing previous entries

Re-fetching updated data

4.4 Editing an Existing Record

Identifying the selected row's ID and updating it correctly required proper mapping between:

TreeView selection

Database entry

4.5 Validations

Preventing empty field submissions and handling invalid input took multiple error checks.

5. Language and Key Takeaways

5.1 Programming Language Used

Python 3 with libraries:

Tkinter (GUI)

Tkcalendar (Date picker)

SQLite3 (Database)

ttk module (TreeView table)

5.2 Key Takeaways

Learned how to build a complete GUI application.

Understood database integration with front-end elements.

Improved program structuring using functions.

Learned how to handle form inputs and data validation.

Gained experience with CRUD operations (Create, Read, Update, Delete).

Enhanced understanding of event-driven programming in Tkinter.

6. Future Enhancements

The application can be improved by adding:

6.1 Expense Categories

Food, Travel, Rent, Shopping, Bills, etc.

6.2 Monthly and Yearly Reports

Automatically calculate:

Total spending

Category-wise breakdown

6.3 Graphs & Charts

Use matplotlib to display:

Pie charts

Bar graphs

Spending trends

6.4 Export Options

Allow user to export data to:

Excel

CSV

PDF

6.5 Authentication System

Add login/logout for user-specific expense tracking.

6.6 Cloud Backup

Sync data with Google Drive or cloud database.

6.7 Mobile App Version

Convert the project into a mobile application using Kivy or Flutter.

7. References

- Python Documentation – <https://docs.python.org>
- Tkinter Official Guide – <https://tkdocs.com>
- SQLite Documentation – <https://sqlite.org/docs.html>

- tkcalendar Library –
<https://github.com/j4321/tkcalendar>
- StackOverflow discussions on Tkinter and TreeView widgets.