Project Report on

EXPENSE TRACKER APP

By
Shivam Batra
Harsh Gulwani
Divyanshu Patil
Manav Puria

Under the Guidance of
Pankti Doshi

Department of Computer Engineering
Mukesh Patel School of Technology, Management, and Engineering

MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT &ENGINEERING
SVKM's

NARSEE MONJEE INSTITUTE OF MANAGEMENT STUDIES
(Declared as Deemed-to-be University Under Section 3 of the UGC Act, 1956)
V. L. Mehta Road, Vile Parle (West)

MUMBAI -400056

March 2020

Subject: Introduction to Programming for Python

Semester III, Year: II

Academic Year: 2024-2025

Program: B.Tech CSDS Division: _____L_____

Sr. No.	Team Members	Roll No.
1	Shivam Batra	L003
2	Harsh Gulwani	L008
3	Divyanshu Patil	L020
4	Manav Puria	L021

Table of Contents

Introduction to Project: 3
What It Does 5
How it Works 6
Data Structure and Suitability 7
Implementation 8
Conclusion 13

Introduction to Project:

This undertaking is an Expense Tracker Application developed by utilizing the tkinter library for the graphical user interface (GUI). The application makes it very easy for the users to keep a record of their income and expenses on a daily and monthly basis with graphical representation to summarize the activities.

What It Does

The Expense Tracker app provides three main tabs:

1. Home (Daily) - Shows total income, expenses, and net balance with an option to add categorized transactions with details.
2. Calendar - Allows selecting a date to view all income and expense transactions for that day.
3. Graph - Displays an annual summary with monthly income, expenses, and net balance as line and bar charts.

The application calculates and displays totals for income and expenses, storing transaction data by date in a dictionary for quick access and aggregation. A built-in calendar picker, customizable transaction categories, and interactive graphs make it easy for users to monitor their financial activity over time

How it Works

This **ExpenseTrackerApp** code creates a Tkinter-based application for tracking finances, organized into three main tabs (Home, Calendar, and Graph). Here's how each part works:

1. **Initialization**:
    a. The application initializes as a Tkinter window titled "Money Tracker" with a custom background and dimensions. It sets up three main tabs (Home, Calendar, and Graph) for different views of the financial data.
    b. It initializes variables for total income, expenses, and a dictionary to store transaction data by date.
2. **Home Tab**:
    a. Displays income, expense, and net balance totals. A button labeled "+" opens a new transaction entry window.
    b. In the transaction window, users can add either income or expenses, choosing categories (like "Salary" or "Food"), inputting the date, amount, category, and notes. The app saves each transaction, updates totals, and stores entries in the dictionary.
3. **Calendar Tab**:
    a. Shows a calendar widget where users can select a date. When a date is selected, it opens a summary window displaying income and expense transactions for that day.
    b. It retrieves and displays transaction details (amount, category, note) for the chosen date from the stored data.
4. **Graph Tab**:
    a. Users select a year, and clicking "Show Summary" triggers an annual summary window. This window aggregates monthly data for the selected year.
    b. Monthly income, expense, and net balance data are displayed as line plots (for income and expenses) and a bar plot (for net balance) using Matplotlib.
5. **Supporting Functions**:
    a. save_transaction(): Adds transactions to the dictionary by date, updating income/expense totals and closing the entry window.
    b. update_totals(): Updates displayed totals on the Home tab.
    c. show_annual_summary(): Calculates monthly income and expenses for the selected year and generates the graphical summary.
    d. open_transaction_summary(): Retrieves and displays transactions for a selected calendar date.

The app continuously updates the Home tab totals and enables easy navigation between daily, monthly, and annual views, providing users with a clear and organized interface to manage finances.

Data Structure and Suitability

The proposed code uses a dictionary to store transaction data by date, where each key is a specific date, and the value is another dictionary with "Income" and "Expense" keys containing lists of transactions. This structure is effective for several reasons:

1. **Efficient Lookup**: It enables quick access to all transactions for a specific day, facilitating daily transaction retrieval in a Calendar view.
2. **Organized Storage**: Separating transactions into "Income" and "Expense" makes categorization and total display easier.
3. **Structured Transactions**: Each transaction is a dictionary with "amount," "category," and "note" keys, providing a clear way to store multiple attributes.

Implementation

The code is explained in brief:

```
import tkinter as tk
from tkinter import ttk, messagebox
from tkcalendar import Calendar, DateEntry
import datetime
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

- tkinter: Python's GUI library for creating graphical user interfaces.
- ttk, messagebox: Widgets and dialogs from tkinter.
- Calendar, DateEntry: Widgets from tkcalendar for calendar date selection.
- datetime: For handling date and time operations.
- matplotlib.pyplot and FigureCanvasTkAgg: Used to plot and embed charts within the Tkinter GUI.

```
class ExpenseTrackerApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Money Tracker")
        self.geometry("400x600")
        self.configure(bg="black")
```

Defines the main application class, ExpenseTrackerApp, which inherits from tk.Tk. This block:

- Sets the window title and dimensions.
- Configures the background color to black.

```
    # Initialize variables to keep track of total income and expenses
    self.income_total = 0.0
    self.expense_total = 0.0
    self.transactions = {}
```

- Initializes variables for total income, total expenses, and a dictionary for storing transaction data.

```
# Create and add the tabs for Daily, Calendar, and Graph
self.daily_tab = ttk.Frame(self.tab_control, style="Black.TFrame")
self.calendar_tab = ttk.Frame(self.tab_control, style="Black.TFrame")
self.graph_tab = ttk.Frame(self.tab_control, style="Black.TFrame")

self.tab_control.add(self.daily_tab, text="Home")
self.tab_control.add(self.calendar_tab, text="Calendar")
self.tab_control.add(self.graph_tab, text="Graph")

self.tab_control.pack(expand=1, fill="both")
```

- Configures a custom ttk style with a black background for the tab frames and creates a Notebook for managing multiple tabs.
- Creates three tabs: daily_tab, calendar_tab, and graph_tab.
- Adds each tab to tab_control and packs the tab control to fill the window

```
# Display labels for income, expense, and total in the Daily tab
self.income_label = tk.Label(self.daily_tab, text="Income: 0.00", fg="cyan", font=("Arial", 14), bg="black")
self.expense_label = tk.Label(self.daily_tab, text="Expense: 0.00", fg="red", font=("Arial", 14), bg="black")
self.total_label = tk.Label(self.daily_tab, text="Net: 0.00", font=("Arial", 14), bg="black", fg="white")

self.income_label.pack(pady=10)
self.expense_label.pack(pady=10)
self.total_label.pack(pady=10)
```

- Displays labels for income, expense, and net balance in the daily_tab.

```
# Button to add new transactions
self.add_transaction_button = tk.Button(self.daily_tab, text="+", font=("Arial", 20), command=self.open_transaction_window, bg="red", fg="white", heigh
self.add_transaction_button.pack(pady=20)
close_button = tk.Button(self.daily_tab, text="Close", command=self.destroy, bg="red", fg="white").pack(pady = 20)
# Set up the Calendar tab for the monthly expense view
self.create_calendar_view()

# Set up the Graph tab for the annual summary view
self.create_graph_view()
```

- Adds a button to open a transaction window for entering new income or expenses.
- Adds a Close button to exit the app.
- Calls the create_calendar_view method to set up the calendar view.
- Calls the create_graph_view method to set up the annual summary view in a chart.

```
def create_calendar_view(self):
    self.calendar = Calendar(self.calendar_tab, selectmode='day', date_pattern='dd/mm/yy')
    self.calendar.pack(pady=20)
    self.calendar.bind("<<CalendarSelected>>", self.open_transaction_summary)
```

- Adds a calendar widget to calendar_tab for selecting dates.
- Binds a callback to open a transaction summary for the selected date.

```
def create_graph_view(self):
    # Year Label and Dropdown Menu
    year_label = tk.Label(self.graph_tab, text="Year:", bg="black", fg="white", font=("Arial", 12))
    year_label.pack(pady=5)

    self.year_combo = ttk.Combobox(self.graph_tab, values=[str(year) for year in range(2000, datetime.datetime.now().year + 1)])
    self.year_combo.pack(pady=5)

    # Show Summary Button
    show_summary_button = tk.Button(self.graph_tab, text="Show Summary", command=self.show_annual_summary, bg="red", fg="white")
    show_summary_button.pack(pady=20)

# Opens a new window to add a transaction
def open_transaction_window(self):
    self.transaction_window = tk.Toplevel(self)
    self.transaction_window.title("Add Transaction")
    self.transaction_window.geometry("400x300")
    self.transaction_window.configure(bg="black")

    # Use tabs to differentiate between Income and Expense entries
    transaction_tabs = ttk.Notebook(self.transaction_window)

    income_tab = ttk.Frame(transaction_tabs, style="Black.TFrame")
    expense_tab = ttk.Frame(transaction_tabs, style="Black.TFrame")

    transaction_tabs.add(income_tab, text="Income")
    transaction_tabs.add(expense_tab, text="Expense")

    transaction_tabs.pack(expand=1, fill="both")
```

- Adds a dropdown for selecting the year and a button to display an annual summary.
- Creates a new window with tabs for adding income and expense entries
- Calls create_transaction_tab to set up input fields for each tab.

```
# Helper function to create input fields for adding a transaction
def create_transaction_tab(self, tab, transaction_type, categories):
    row = 0

    # Input for date with calendar dropdown
    date_label = tk.Label(tab, text="Date:", bg="black", fg="white")
    date_label.grid(row=row, column=0, sticky="w", padx=10, pady=5)
    date_entry = DateEntry(tab, date_pattern='dd/mm/yy', background="black", foreground="white")
    date_entry.grid(row=row, column=1, sticky="w", padx=10, pady=5)
    row += 1
    # Input for amount
    amount_label = tk.Label(tab, text="Amount (Rupees):", bg="black", fg="white")
    amount_label.grid(row=row, column=0, sticky="w", padx=10, pady=5)
    amount_entry = tk.Entry(tab)
    amount_entry.grid(row=row, column=1, sticky="w", padx=10, pady=5)

    row += 1
    # Dropdown for category
    category_label = tk.Label(tab, text="Category:", bg="black", fg="white")
    category_label.grid(row=row, column=0, sticky="w", padx=10, pady=5)
    category_combo = ttk.Combobox(tab, values=categories)
    category_combo.grid(row=row, column=1, sticky="w", padx=10, pady=5)

    row += 1
    # Note entry
    note_label = tk.Label(tab, text="Note:", bg="black", fg="white")
    note_label.grid(row=row, column=0, sticky="w", padx=10, pady=5)
    note_entry = tk.Entry(tab)
    note_entry.grid(row=row, column=1, sticky="w", padx=10, pady=5)

    row += 1
    # Save button
    save_button = tk.Button(tab, text="Save", bg="red", fg="white", command=lambda: self.save_transaction(transaction_type, date_entry.get(), amount_entry.
    save_button.grid(row=row, column=1, sticky="e", padx=10, pady=10)
    close_button = tk.Button(tab, text="Close", command=self.transaction_window.destroy, bg="red", fg="white").grid(row=row, column=3, sticky="e", padx=10,
```

- Sets up input fields for date, amount, category, and a note.
- Adds a Save button to store transaction data.

```
def show_annual_summary(self):
    selected_year = self.year_combo.get()
    if not selected_year:
        messagebox.showerror("Error", "Please select a year")
        return

    # Initialize monthly income and expense data to zero
    income_data = [0] * 12
    expense_data = [0] * 12

    # Aggregate data from transactions
    for date, entry_types in self.transactions.items():
        transaction_date = datetime.datetime.strptime(date, '%d/%m/%y')
        if transaction_date.year == int(selected_year):
            month_index = transaction_date.month - 1
            for entry_type, entries in entry_types.items():
                for entry in entries:
                    if entry_type == "Income":
                        income_data[month_index] += entry["amount"]
                    elif entry_type == "Expense":
                        expense_data[month_index] += entry["amount"]

    net_data = [income - expense for income, expense in zip(income_data, expense_data)]
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

    annual_summary_window = tk.Toplevel(self)
    annual_summary_window.title("Annual Summary")
    annual_summary_window.geometry("700x900")
    annual_summary_window.configure(bg="black")

    tk.Label(annual_summary_window, text=f"Year: {selected_year}", font=("Arial", 14), bg="black", fg="white").pack(pady=10)

    # Create a single figure with Income and Expense as line plots and Net as a bar plot
    fig, ax = plt.subplots(figsize=(8, 8), dpi=80)
    fig.patch.set_facecolor("black")
```

- Retrieves data for the selected year, calculates monthly totals, and displays an annual summary chart in a new window.

```
def open_transaction_summary(self, event):
    selected_date = self.calendar.get_date()
    transaction_summary_window = tk.Toplevel(self)
    transaction_summary_window.title(f"Transactions on {selected_date}")
    transaction_summary_window.geometry("300x300")
    transaction_summary_window.configure(bg="black")

    if selected_date in self.transactions:
        income_transactions = self.transactions[selected_date].get("Income", [])
        expense_transactions = self.transactions[selected_date].get("Expense", [])

        tk.Label(transaction_summary_window, text="Income", fg="blue", bg="black", font=("Arial", 12, "bold")).pack(pady=5)
        for transaction in income_transactions:
            tk.Label(transaction_summary_window, text=f"{transaction['category']}: {transaction['amount']} (Note: {transaction['note']})", bg="black", fg="

        tk.Label(transaction_summary_window, text="Expense", fg="red", bg="black", font=("Arial", 12, "bold")).pack(pady=5)
        for transaction in expense_transactions:
            tk.Label(transaction_summary_window, text=f"{transaction['category']}: {transaction['amount']} (Note: {transaction['note']})", bg="black", fg="
    else:
        tk.Label(transaction_summary_window, text="No transactions for this date.", bg="black", fg="white").pack(pady=10)

def save_transaction(self, transaction_type, date, amount, category, note):
    try:
        amount = float(amount)
    except ValueError:
        messagebox.showerror("Error", "Invalid amount")
        return

    # Save transaction data
    if date not in self.transactions:
        self.transactions[date] = {"Income": [], "Expense": []}

    self.transactions[date][transaction_type].append({"amount": amount, "category": category, "note": note})
```

- Displays a list of transactions for the selected date, split into income and expenses.
- Validates and saves the transaction in self.transactions.
- Updates the income or expense total based on the transaction type and refreshes the totals display.

```
    def update_totals(self):
        self.income_label.config(text=f"Income: {self.income_total:.2f}")
        self.expense_label.config(text=f"Expense: {self.expense_total:.2f}")
        total = self.income_total - self.expense_total
        self.total_label.config(text=f"Net: {total:.2f}")

# Run the application
app = ExpenseTrackerApp()
app.mainloop()
```

- Updates the display labels for income, expenses, and net balance in daily_tab.
- Instantiates and runs the ExpenseTrackerApp, displaying the interface and handling events.

Conclusion

This expense tracker app, built with Python's Tkinter library, offers an organized interface for managing personal finances by recording income and expenses. Users can view financial summaries in three tabs: a daily view, a calendar-based view, and a graphical annual summary. The app provides an intuitive form for entering transactions, complete with date, category, amount, and notes, and stores these entries to calculate monthly and yearly financial data. Through this system, users can gain insights into their spending habits and track their net balance over time, all within an easy-to-navigate, interactive interface.