Prepared For ICSS

EXPENSE TRACKER APPLICATION

12-10-2023

Presented To
PIJUSH SINGH

Presented By
SAMIUL TOMAL

Building an Expense Tracker Application
Using Python and tkinter

Puropse:

The purpose of this documentation is to introduce and explain the development of an Expense Tracker application created using Python and the Tkinter library. I will delve into the application's code structure, its various functions, and its graphical user interface components. By the end of this presentation, you'll have a clear understanding of how this Expense Tracker application works and how it can be used to manage personal or business expenses.

What is tkinter?

Tkinter is a Python library used for creating graphical user interfaces (GUIs). It provides a set of tools and functions that allow developers to design and build user-friendly applications with windows, buttons, text fields, and other interactive elements. Tkinter is a standard library that comes bundled with most Python installations, making it readily available and widely used for developing desktop applications.

Tkinter as a Python GUI Library

Tkinter serves as Python's standard GUI library, and it enables developers to design cross-platform desktop applications with ease. It provides a range of widgets and layout options for building interactive applications, making it a go-to choice for creating user interfaces in Python.

Popularity and Ease of Use

Popularity: Tkinter is highly popular among Python developers due to its inclusion in the standard library. This means that it's accessible to anyone using Python, and there's a large community of users and resources available for support.

Ease of Use: Tkinter is known for its simplicity and ease of use. It has a straightforward and intuitive design, which makes it a great choice for both beginners and experienced developers. Creating a basic GUI in Tkinter often requires only a few lines of code.

The Expense Tracker Application

The Expense Tracker application is the central focus of our presentation. This application has been developed using Python and the tkinter library. It serves a valuable purpose in helping users manage their expenses efficiently.

Purpose and Design

The Expense Tracker application is designed to simplify the process of recording and monitoring expenses. It provides users with a user-friendly and interactive platform to:

Add Expenses: Users can input details of their expenses, including the expense name, date, and the amount spent.

Search Expenses: The application allows users to search for specific expenses based on keywords or criteria, making it easy to retrieve and review past transactions.

Export to Excel: Users can export their expense records to an Excel file for further analysis and record-keeping.

Clear Entries: The application provides options to clear individual entries or all expense records, offering flexibility in managing data.

Code Structure

Importing Libraries:
At the beginning of the code, several Python libraries are imported. These libraries provide essential tools and functionalities for the application:
tkinter: Used to create the graphical user interface (GUI) components.
messagebox: Enables the display of pop-up message boxes for user interaction.
openpyxl: Provides the necessary tools to work with Excel files. This library is used for exporting expenses to an Excel file.

Defining Functions:
The code defines several functions to handle various aspects of the application's functionality. These functions are responsible for:

Adding expenses to the list.
Updating the displayed list of expenses.
Clearing individual entry fields.
Clearing all entries and the expenses list.
Searching for expenses based on user input.
Exporting expenses to an Excel file.

Creating the Main tkinter Window:

The main application window is created using the tk.Tk() class. This window is the primary container for all the GUI elements and serves as the user interface for the application.

Setting up Labels, Entry Fields, Buttons, and the Listbox:

Within the main window, various GUI elements are set up, including:

Labels: These provide descriptions for the input fields and buttons.
Entry Fields: These are used for users to input expense details such as name, date, and amount.
Buttons: These allow users to trigger actions, such as adding expenses, searching, exporting, and clearing entries.

Listbox: This component is used to display the list of expenses on the application's screen, making it easy for users to view and manage their recorded expenses.

Event Handling Functions:

Event handling functions are associated with the buttons and are called when specific user actions occur. For example:

The **add_expenses()** function is called when the "Add Expense" button is clicked.
The **search_expenses()** function is triggered when the "Search" button is pressed.
The **export_expenses_excel()** function is executed when the "Export to Excel" button is clicked.

Function: add_expenses()

```
def add_expenses():
    # Retrieve user input from the name, date, and amount Entry fields.
    name = name_entry.get()
    date = date_entry.get()
    amount = amount_entry.get()

    # Check if all required fields are filled in.
    if name and date and amount:
        # Create a tuple with the expense details and add it to the expenses_list.
        expenses_list.append((name, date, amount))
        # Update the displayed list of expenses.
        update_expenses()
        # Clear the input fields.
        clear_entries()
    else:
        # If any of the fields are empty, display an error message.
        messagebox.showerror("Error", "Please fill in all fields.")
```

Explanation of the Code:

The add_expenses() function starts by retrieving user input from the name, date, and amount Entry fields using the .get() method. This input is stored in the name, date, and amount variables.

It then checks if all the required fields are filled in. This is done by verifying if name, date, and amount variables are not empty (i.e., they have some content).

If all fields are filled in, a tuple containing the expense details (name, date, and amount) is created. This tuple is then appended to the expenses_list, which stores all the expense records.

After adding the new expense, the function calls update_expenses() to refresh the displayed list of expenses in the GUI, ensuring that the newly added expense is included.

Finally, the clear_entries() function is called to clear the input fields, preparing them for the next expense entry.

If any of the required fields are empty, the function uses the messagebox.showerror() method to display an error message, alerting the user to fill in all the fields before adding an expense.

Function: update_expenses()

```
def update_expenses():
    # Clear the existing list of expenses in the GUI.
    expenses_box.delete(0, tk.END)

    # Iterate through the expenses_list and insert each expense as a formatted string into the
GUI list.
    for i, expense in enumerate(expenses_list, start=1):
        formatted_expense = f'{i}. Name: {expense[0]}, Date: {expense[1]}, Amount:
{expense[2]}'
        expenses_box.insert(tk.END, formatted_expense)
```

Explanation of the Code:

The update_expenses() function begins by clearing the existing list of expenses in the GUI. This is done using the expenses_box.delete(0, tk.END) method, which removes all items (lines) in the expenses_box Listbox widget.

It then proceeds to iterate through the expenses_list, which contains all the recorded expenses. The enumerate(expenses_list, start=1) function is used to loop through the list while keeping track of the index, starting from 1.

For each expense in the list, a formatted string is created, including the expense's index, name, date, and amount. This string is stored in the formatted_expense variable.

Finally, the formatted_expense string is inserted into the expenses_box Listbox widget using the expenses_box.insert(tk.END, formatted_expense) method. This action populates the GUI with the updated list of expenses.

Function: clear_entries()

```
def clear_entries():
    # Use the delete(0, tk.END) method to clear the input fields.
    name_entry.delete(0, tk.END)
    date_entry.delete(0, tk.END)
    amount_entry.delete(0, tk.END)
```

Explanation of the Code:

The clear_entries() function uses the .delete(0, tk.END) method on each of the input fields (name, date, and amount) to clear their contents.

name_entry.delete(0, tk.END) clears the name Entry field by deleting all content within it, from the start (0) to the end (tk.END).

Similarly, date_entry.delete(0, tk.END) and amount_entry.delete(0, tk.END) clear the date and amount Entry fields in the same manner.

Function: clear_entries()

```
def clear_entries():
    # Use the delete(0, tk.END) method to clear the input fields.
    name_entry.delete(0, tk.END)
    date_entry.delete(0, tk.END)
    amount_entry.delete(0, tk.END)
```

Explanation of the Code:

The clear_entries() function uses the .delete(0, tk.END) method on each of the input fields (name, date, and amount) to clear their contents.

name_entry.delete(0, tk.END) clears the name Entry field by deleting all content within it, from the start (0) to the end (tk.END).

Similarly, date_entry.delete(0, tk.END) and amount_entry.delete(0, tk.END) clear the date and amount Entry fields in the same manner.

Function: clear_all_entries()

```
def clear_all_entries():
    # Clear all input fields for name, date, and amount.
    name_entry.delete(0, tk.END)
    date_entry.delete(0, tk.END)
    amount_entry.delete(0, tk.END)

    # Clear the expenses_list by removing all elements.
    expenses_list.clear()

    # Update the displayed list of expenses (GUI) to reflect the changes.
    update_expenses()
```

Explanation of the Code:

The clear_all_entries() function starts by using the .delete(0, tk.END) method on each of the input fields (name, date, and amount) to clear their contents, ensuring that all entry fields are empty.

Next, the function clears the expenses_list, which stores all the recorded expenses. It does so by using the .clear() method on the list, effectively removing all elements.

After clearing both the input fields and the expenses list, the function ensures that the displayed list of expenses in the GUI is updated to reflect the changes. This is achieved by calling the update_expenses() function, which repopulates the list of expenses in the user interface.

Function: search_expenses()

```
def search_expenses():
    # Retrieve the search text entered by the user, converting it to lowercase for
case-insensitive search.
    search_text = search_entry.get().lower()

    # Create a list to store the search results.
    results = []

    # Iterate through the expenses_list and check if any expense matches the search criteria.
    for expense in expenses_list:
        if (search_text in expense[0].lower() or
            search_text in expense[1].lower() or
            search_text in expense[2].lower()):
            results.append(expense)

    # Clear the displayed list of expenses in the GUI.
    expenses_box.delete(0, tk.END)

    # Display the search results in the GUI.
    for i, expense in enumerate(results, start=1):
        formatted_expense = f'{i}. Name: {expense[0]}, Date: {expense[1]}, Amount:
{expense[2]}'
        expenses_box.insert(tk.END, formatted_expense)
```

Explanation of the Code:

The search_expenses() function starts by retrieving the search text entered by the user from the search_entry input field. It converts the search text to lowercase to enable case-insensitive search.

A list called results is created to store the search results, which are the expenses that match the search criteria.

The function then iterates through the expenses_list, checking each expense to see if it matches the search criteria. It performs the search by comparing the lowercase version of the search text with the lowercase versions of the expense name, date, and amount.

Expenses that match the search criteria are appended to the results list.

Before displaying the search results, the function clears the existing list of expenses in the GUI using expenses_box.delete(0, tk.END).

Finally, the function inserts the search results into the expenses_box Listbox widget, updating the displayed list to show the expenses that match the user's search criteria.

Function: export_expenses_excel()

```
def export_expenses_excel():
    # Create a new Excel workbook and select the active worksheet.
    wb = Workbook()
    ws = wb.active

    # Add a header row to the worksheet.
    ws.append(["Name", "Date", "Amount"])

    # Iterate through the expenses_list and add each expense as a row in the worksheet.
    for expense in expenses_list:
        ws.append(expense)

    # Save the Excel workbook to a file named "Expenses.xlsx."
    wb.save('Expenses.xlsx')

    # Display a message to inform the user that the export is complete.
    messagebox.showinfo('Export', "Expenses Exported to Expenses.xlsx")
```

Explanation of the Code:

The export_expenses_excel() function begins by creating a new Excel workbook using the Workbook() constructor from the openpyxl library. It also selects the active worksheet in the workbook, which is the default worksheet named "Sheet."

A header row is added to the worksheet using the ws.append(["Name", "Date", "Amount"]) method. This row contains column headers for "Name," "Date," and "Amount."

The function then iterates through the expenses_list, which contains all recorded expenses, and adds each expense as a row in the worksheet. This effectively transfers the expense data from the application to the Excel worksheet.

After all expenses are added to the worksheet, the Excel workbook is saved to a file named "Expenses.xlsx" using the wb.save('Expenses.xlsx') method. The user's expense records are now stored in this Excel file.

Finally, a message box is displayed using messagebox.showinfo(), informing the user that the export process is complete and that the expenses have been successfully exported to the "Expenses.xlsx" file.

The Graphical User Interface (GUI)

The graphical user interface (GUI) of the Expense Tracker application is designed to provide users with a user-friendly and intuitive platform for managing their expenses. Below is a visual representation of the GUI, along with explanations of the purpose of each interface element:

1. Labels:

"Expenses Name": This label indicates the purpose of the adjacent entry field and instructs the user to input the name of the expense.
"Date": This label specifies the adjacent entry field for entering the date of the expense.
"Amount": This label identifies the adjacent entry field for entering the amount spent.

2. Entry Fields:

Name Entry Field: This is where the user enters the name or description of the expense.
Date Entry Field: Users input the date of the expense in this field.
Amount Entry Field: The user specifies the amount spent in this field.

3. Buttons:

"Add Expense": Clicking this button triggers the add_expenses() function, which adds the expense to the list and updates the displayed expenses.
"Search": This button activates the search_expenses() function, enabling users to search for specific expenses based on their input criteria.
"Export to Excel": Clicking this button initiates the export_expenses_excel() function, which exports the expense records to an Excel file.
"Clear All": Clicking this button calls the clear_all_entries() function, clearing all input fields and removing all recorded expenses.

4. Search Entry Field:

This entry field is specifically provided for users to enter search criteria when using the "Search" button to find expenses based on specific keywords.

5. Listbox:

The Listbox widget displays the list of expenses in the GUI. It updates dynamically to reflect the changes made to the expenses list. Each expense is listed with its name, date, and amount.

Running the Application

The Expense Tracker application can be executed in a Python environment, allowing users to manage their expenses efficiently. To run the application successfully, there are a few requirements and dependencies to keep in mind:

1. Python Environment:

Ensure that you have a working Python environment installed on your system. The Expense Tracker application is written in Python and requires Python to run.

2. Required Libraries:

The application relies on two specific libraries, so make sure they are installed:
tkinter: As a part of the standard library, tkinter should be readily available with your Python installation. If it's not, you may need to install it separately.
openpyxl: To handle Excel file operations, you'll need to install the openpyxl library. You can install it using pip with the following command:

pip install openpyxl

Once you have Python installed and the required libraries set up, you can run the Expense Tracker application by executing the Python script. This will launch the graphical user interface, allowing you to add, search, export, and manage your expenses effectively.

Keep in mind that the application's GUI and functionalities, as well as its ease of use, make it a powerful tool for tracking expenses, whether for personal finance management or for business purposes.

Demonstration

In this section, we'll walk you through how to use the Expense Tracker application effectively. We'll cover the essential actions you can perform with the application, including adding expenses, searching, exporting to Excel, and clearing entries.

1. Adding Expenses:

Start by filling in the "Expenses Name," "Date," and "Amount" entry fields with the details of your expense.
Click the "Add Expense" button to add the expense to the list.
The expense will be displayed in the Listbox on the GUI.

2. Searching Expenses:

To search for a specific expense, enter keywords or criteria related to the expense in the "Search" entry field.
Click the "Search" button.
The Listbox will be updated to display only the expenses that match your search criteria.

3. Exporting to Excel:

If you wish to export your expenses to an Excel file for further analysis, click the "Export to Excel" button.
The application will generate an Excel file named "Expenses.xlsx" containing your expense records.

4. Clearing Entries:

If you want to clear all the input fields and remove all recorded expenses, click the "Clear All" button.
This action will clear the input fields and reset the application to its initial state.
Additional Tips:

To clear individual entries, you can use the "Clear" button for each field.
The Listbox updates dynamically as you add, search, or clear expenses, providing a real-time view of your financial transactions.

The Expense Tracker application simplifies expense management by providing a user-friendly interface for entering, searching, and tracking expenses. Whether you're managing personal finances or business expenditures, this tool makes it easy to stay organized and informed about your expenses.

Conclusion

In this Documentation, we have explored the Expense Tracker application, a user-friendly tool designed to help individuals and businesses manage their expenses effectively. Let's recap the key points and highlight the significance of this application:

Expense Tracker Purpose: The Expense Tracker application is designed to streamline the process of recording and monitoring expenses. It offers features for adding expenses, searching, exporting to Excel, and clearing entries.

Code Overview: We examined the code structure, which is divided into sections for importing libraries, defining functions, creating the main tkinter window, setting up GUI components, and handling user events.

Functions: The presentation explained essential functions like adding expenses, updating the displayed list, clearing entries, clearing all entries, searching expenses, and exporting to Excel.

The GUI: We provided a visual representation of the graphical user interface, detailing the purpose of labels, entry fields, buttons, and the Listbox. Each element plays a role in making expense management efficient.

Running the Application: To run the Expense Tracker application, you need a Python environment with tkinter and openpyxl installed. This tool offers a straightforward way to manage expenses, whether for personal finance or business.

Demonstration: We demonstrated how to use the application, covering actions such as adding expenses, searching for specific records, exporting expenses to Excel, and clearing entries.

The Importance of the Expense Tracker Application:

The Expense Tracker application is a valuable tool for anyone looking to maintain financial discipline, track expenses, and gain insights into their spending patterns. It simplifies the process of recording and managing expenses, offering a user-friendly interface and practical features for effective expense management. Whether you're striving to budget more efficiently, monitor business expenditures, or simply stay organized with your finances, the Expense Tracker application is a practical and accessible solution.

With its easy-to-use GUI and essential functionalities, the Expense Tracker application empowers users to take control of their expenses, ultimately leading to better financial management and informed decision-making. Its capability to export data to Excel further enhances its utility, allowing users to analyze their spending trends and make informed financial choices.

In conclusion, the Expense Tracker application offers a powerful and user-friendly solution for managing expenses and gaining control over your financial well-being. Whether you are an individual or a business owner, this tool can be a significant asset in your financial management toolkit.

Thank you