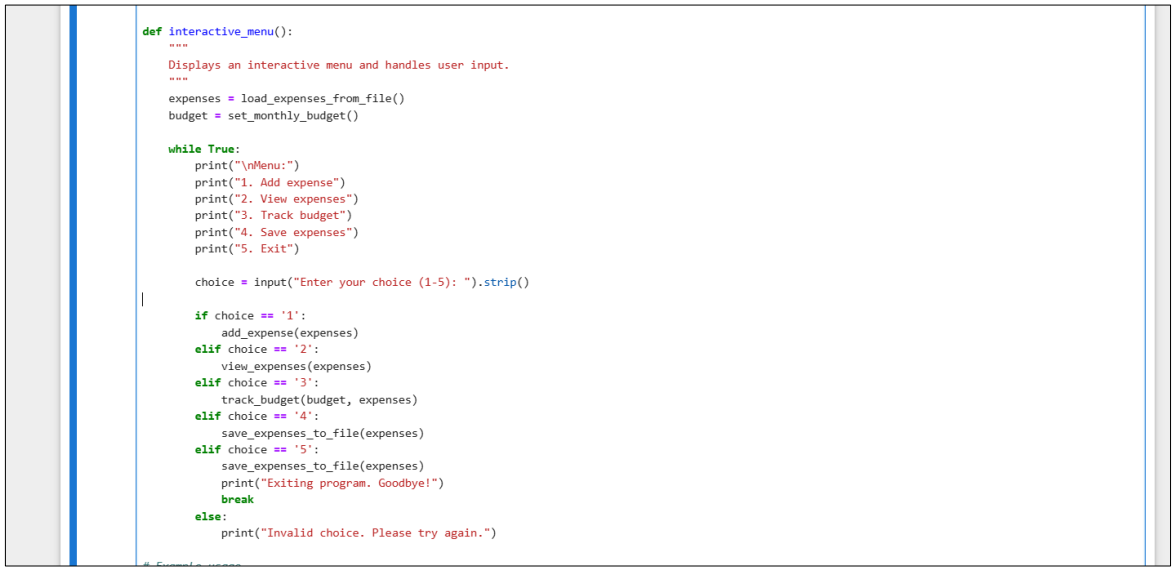
Personal Expense Tracker

Writeup

Let's start from main function of the source code i.e., from where the execution of the code starts:

Step 1:

Here the main function starts by executing the **interactive_menu()** of the source code. The function starts by asking the user their monthly budget by executing **set_monthly_budget() a**nd loads expenses from the expenses.csv file **load_expenses_from_file()**.



```
def interactive_menu():
    """
    Displays an interactive menu and handles user input.
    """
    expenses = load_expenses_from_file()
    budget = set_monthly_budget()

    while True:
        print("\nMenu:")
        print("1. Add expense")
        print("2. View expenses")
        print("3. Track budget")
        print("4. Save expenses")
        print("5. Exit")

        choice = input("Enter your choice (1-5): ").strip()

        if choice == '1':
            add_expense(expenses)
        elif choice == '2':
            view_expenses(expenses)
        elif choice == '3':
            track_budget(budget, expenses)
        elif choice == '4':
            save_expenses_to_file(expenses)
        elif choice == '5':
            save_expenses_to_file(expenses)
            print("Exiting program. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
```

```
def load_expenses_from_file(filename="expenses.csv"):
    """
    Loads expenses from a CSV file.
    :param filename: Name of the file to load expenses from.
    :return: List of expenses loaded from the file.
    """
    try:
        with open(filename, mode='r') as file:
            reader = csv.DictReader(file)
            expenses = [
                {
                    'date': row['date'],
                    'category': row['category'],
                    'amount': float(row['amount']),
                    'description': row['description']
                } for row in reader
            ]
            print(f"Expenses loaded from {filename}\n")
            return expenses
    except FileNotFoundError:
        print(f"No existing expenses file found. Starting fresh.\n")
        return []
    except Exception as e:
        print(f"Error loading expenses from file: {e}")
        return []
```

```python
def set_monthly_budget():
    """
    Prompts the user to set a monthly budget.
    :return: The budget amount as a float.
    """
    try:
        budget = float(input("Enter your monthly budget: ").strip())
        print(f"Monthly budget set to ${budget:.2f}\n")
        return budget
    except ValueError:
        print("Invalid input. Please enter the budget as a numeric value.")
```

Step 2:

From the interactive menu, based on the choice selected by the user the further execution happens

For Example: If the user selects choice '1', the **add_expense()** will be called and expenses will be added to a list.

```python
def add_expense(expenses_list):
    """
    Prompts the user to input expense details and stores them as a dictionary in a given list.
    :param expenses_list: List to store expense dictionaries.
    """
    try:
        # Prompt user for expense details
        date = input("Enter the date of the expense (YYYY-MM-DD): ").strip()
        category = input("Enter the category of the expense: ").strip()
        amount = float(input("Enter the amount spent: ").strip())
        description = input("Enter a brief description of the expense: ").strip()

        # Validate the date format
        from datetime import datetime
        try:
            datetime.strptime(date, "%Y-%m-%d")
        except ValueError:
            print("Invalid date format. Please use YYYY-MM-DD.")
            return

        # Create an expense dictionary
        expense = {
            'date': date,
            'category': category,
            'amount': amount,
            'description': description
        }

        # Add the expense to the list
        expenses_list.append(expense)
        print("Expense added successfully!\n")

    except ValueError:
        print("Invalid input. Please enter the amount as a numeric value.")
```

Step 3:

Similarly if choice 2 is chosen, view_expenses(), all the expenses added to expenses.csv file will be displayed

```python
def view_expenses(expenses_list):
    """
    Displays all stored expenses in a formatted manner.
    :param expenses_list: List containing expense dictionaries.
    """
    if not expenses_list:
        print("No expenses to display.")
        return

    print("\nStored Expenses:")
    for expense in expenses_list:
        # Validate expense data
        if not all(key in expense for key in ('date', 'category', 'amount', 'description')):
            print("Incomplete expense entry found and skipped.")
            continue

        print(f"Date: {expense['date']}")
        print(f"Category: {expense['category']}")
        print(f"Amount: ${expense['amount']:.2f}")
        print(f"Description: {expense['description']}")
        print("-" * 30)
```

Step 4:

If the user chooses option 3, **track_budget()** will be executed first.

At first we will get the total expenses from **calculate_total_expenses(**). If the total expenses is more than the budget the user has the function returns a warning message else the remaining balance budget amount is returned.

```
def calculate_total_expenses(expenses_list):
    """
    Calculates the total expenses recorded so far.
    :param expenses_list: List containing expense dictionaries.
    :return: Total amount of expenses as a float.
    """
    return sum(expense.get('amount', 0) for expense in expenses_list)

def track_budget(budget, expenses_list):
    """
    Compares the total expenses with the monthly budget and displays the status.
    :param budget: The monthly budget amount.
    :param expenses_list: List containing expense dictionaries.
    """
    total_expenses = calculate_total_expenses(expenses_list)
    print(f"Total expenses so far: ${total_expenses:.2f}")

    if total_expenses > budget:
        print("Warning: You have exceeded your budget!")
    else:
        remaining = budget - total_expenses
        print(f"You have ${remaining:.2f} left for the month.\n")
```

Step 5:

If the user chooses option 4, **save_expenses_to_file()** will be executed. Here, the expenses will be saved to the expenses.csv file from the List to each data being considered as a dictionary. The **DictWriter** lets you write CSV files very neatly and semantically by defining each row as a Python dict.

```
def save_expenses_to_file(expenses_list, filename="expenses.csv"):
    """
    Saves all expenses to a CSV file.
    :param expenses_list: List containing expense dictionaries.
    :param filename: Name of the file to save expenses to.
    """
    try:
        with open(filename, mode='w', newline='') as file:
            writer = csv.DictWriter(file, fieldnames=['date', 'category', 'amount', 'description'])
            writer.writeheader()
            writer.writerows(expenses_list)
        print(f"Expenses saved to {filename}\n")
    except Exception as e:
        print(f"Error saving expenses to file: {e}")
```

Step 6: Last and the final choice is to finally again save the data into CSV file by executing **save_expenses_to_file()** and exit the program if the user chooses choice 5