```
import csv
```

# To load a built-in library in Python to handle CSV files. So that expenses can be saved to and loaded from a file.

```
expenses = []
monthly_budget = 0
```

# Initializes an empty list to store expense entries.

# Sets the initial monthly budget to zero.

## 1. Add an Expenses

```
def add_expense():
    date = input("Enter date (YYYY-MM-DD): ")
    category = input("Enter category (e.g., Food, Travel): ")
    amount = float(input("Enter amount: "))
    description = input("Enter description: ")
    expenses.append({"date": date,"category": category,"amount": amount,"description": description
})
    print("Expense added.\n")
```

In the above code,

**def add_expense():** #-> This function takes details from users to add a new expense and add it to the expenses list.

**date = input("Enter date (YYYY-MM-DD): ")**

**category = input("Enter category (e.g., Food, Travel): ")**

**amount = float(input("Enter amount: "))**

**description = input("Enter description: ")**

#-> To get the date, category, amount, and description of an expense from the user.

**expenses.append({"date": date,"category": category,"amount": amount,"description": description})** **#->** created a dictionary for the received details and added to the expense list.

**print("Expense added.\n")** **#->** Notifying the user that the expense was successfully added.

**\n** **#->** for new line

## 2. View Expenses

```
def view_expenses():
    if not expenses:
        print("No expenses recorded.\n")
        return
    print("Your Expenses:")
    for exp in expenses:
        if all(k in exp for k in ("date", "category", "amount", "description")):
            print(f'{exp["date"]} | {exp["category"]} | ₹{exp["amount"]} | {exp["description"]}')
    print()
```

**def view_expenses():** #-> Function to view a list of all expenses.

**if not expenses:**

**print("No expenses recorded.\n")**

**return**

#-> If the expenses list is empty, prints a message and returns.

**print("Your Expenses:")**

**for exp in expenses:**

**if all(k in exp for k in ("date", "category", "amount", "description")):**

**print(f'{exp["date"]} | {exp["category"]} | ₹{exp["amount"]} | {exp["description"]}')**

#-> Otherwise, prints each expense's details in a formatted manner. Here, **f** is formatted string literals

**print()** #-> Leave a blank line below.

## 3. Set Budget

```
def set_budget():
    global monthly_budget
    monthly_budget = float(input("Set your monthly budget: ₹"))
    print("Budget set.\n")
```

**def set_budget(): #->** This function lets the user set the monthly budget.

**global monthly_budget** #-> Uses the global variable so its value can be accessed and modified throughout the program.

**monthly_budget = float(input("Set your monthly budget: ₹"))** #-> Storing the user's budget in a global variable. Here, input()=user input, float()=typecasting,

**print("Budget set.\n")** #-> Prints a confirmation message.

## 3. Track Budget

```
def track_budget():
    total_expense = sum(exp['amount'] for exp in expenses)
    print(f"Total expenses so far: ₹{total_expense}")
    if total_expense > monthly_budget:
        print("Warning: You have exceeded your budget!")
    else:
        print(f"Remaining budget: ₹{monthly_budget - total_expense}")
    print()
```

**def track_budget():** #-> this function to show expenses and budget status.

**total_expense = sum(exp['amount'] for exp in expenses)**

**print(f"Total expenses so far: ₹{total_expense}")**

#-> Calculate the total expenses by summing the amount from each expense and displays the total spend

**if total_expense > monthly_budget:**

**print("Warning: You have exceeded your budget!")**

**else:**

**print(f"Remaining budget: ₹{monthly_budget - total_expense}")**

#-> Compare with the monthly budget and prints whether the user is over budget or how much is left.

**print()** #-> Leave a blank line below.

## 4. Save Expenses

```
def save_expenses():
    with open("expenses.csv", "w", newline='', encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=["date", "category", "amount", "description"])
        writer.writeheader()
        for exp in expenses:
```

```
        writer.writerow(exp)
    print("Expenses saved to CSV.\n")
```

**def save_expenses():** #-> this function to save all expense information.

**with open("expenses.csv", "w", newline='', encoding="utf-8") as f:**

#-> Opens a file called 'expenses.csv' for writing. Here, **newline=''** = to type column wise line in CSV file, use a loop while to type row wise lines, **encoding="utf-8"** = To avoid errors or misread data in the file so that data can be saved in a different language, 'w' = File Handling Modes for write document, as = alias

**writer = csv.DictWriter(f, fieldnames=["date", "category", "amount", "description"])**

#-> Sets up a CSV writer to write dictionaries with specified fields.

**writer.writeheader()** #-> Writes the headers in the file.

**for exp in expenses:**

**writer.writerow(exp)**

#-> for loop has been used to write each expense entry as a row in the file..

**print("Expenses saved to CSV.\n")** #-> Prints a confirmation message.

## 4. Load Expenses

```
def load_expenses():
    try:
        with open("expenses.csv", "r", encoding="utf-8") as f:
            reader = csv.DictReader(f)
            for row in reader:
                expenses.append({
                    "date": row["date"],
                    "category": row["category"],
                    "amount": float(row["amount"]),
                    "description": row["description"]
                })
        print("Previous expenses loaded.\n")
    except FileNotFoundError:
```

```
    print("No previous expense data found.\n")
```

**def load_expenses():** #-> this function to load expenses from a CSV file before the program starts.

```
  try:
    with open("expenses.csv", "r", encoding="utf-8") as f:
      reader = csv.DictReader(f)
      for row in reader:
        expenses.append({
          "date": row["date"],
          "category": row["category"],
          "amount": float(row["amount"]),
          "description": row["description"]
        })
    print("Previous expenses loaded.\n")
```

#-> If file is found, reads each row and adds it to the expenses list. Here, '**r**'= File Handling Modes for read/load the document

```
except FileNotFoundError:
    print("No previous expense data found.\n")
```

#-> If file not found, prints that no previous data is available

## 5. Create an Interactive Menu

```
def menu():
  load_expenses()
  while True:
    print("--- Menu ---")
    print("1. Add Expense")
    print("2. View Expenses")
    print("3. Set Budget")
    print("4. Track Budget")
    print("5. Save Expenses")
    print("6. Exit")
    choice = input("Choose an option (1-6): ")
```

```
    if choice == '1': add_expense()
    elif choice == '2': view_expenses()
    elif choice == '3': set_budget()
    elif choice == '4': track_budget()
    elif choice == '5': save_expenses()
    elif choice == '6':
        save_expenses()
        print("Exiting the program.")
        break
    else:
        print("Invalid option.\n")
```

**def menu():** #-> The main menu function.

**load_expenses()** #-> Loads previously saved expenses before presenting options to the user.

```
  while True:
    print("--- Menu ---")
    print("1. Add Expense")
    print("2. View Expenses")
    print("3. Set Budget")
    print("4. Track Budget")
    print("5. Save Expenses")
    print("6. Exit")
    choice = input("Choose an option (1-6): ")
```

#-> Repeatedly displays menu options and lets the user select what to do.

```
    if choice == '1': add_expense()
    elif choice == '2': view_expenses()
    elif choice == '3': set_budget()
    elif choice == '4': track_budget()
    elif choice == '5': save_expenses()
    elif choice == '6':
        save_expenses()
```

**print("Exiting the program.")**

**break**

#-> Based on choice, calls the appropriate function. If the user chooses to exit, saves data and ends the program.

**else:**

**print("Invalid option.\n")**

#-> If the user selected option is outside the menu options, the selected option is invalid.

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

**if __name__ == "__main__":** #->This condition checks if the script is being run directly.

**menu()** #->It then starts the program by calling the menu function.