Technical Design Document

Name: Elvin Karimov

Date Created: June 13, 2025

Program Description:

This program asks the user for a list of their monthly expenses, including the type and amount of each expense, and then uses a reduced method from the functools module to look at them. It figures out the total costs, finds the highest and lowest costs, and gives each one a label based on its type. The program makes sure that input validation is strong enough to handle amounts that aren't numbers and that at least one expense entry is required. It talks about the need to look at and summarize financial data, giving clear information about spending habits for budgeting.

Functions Used in the Program (Listed in Order as They Are Called):

1. Function Name: collect\_expenses

Description: Prompts the user to enter expense types and amounts, validates inputs, and returns a list of expense dictionaries.

Parameters: None

Variables:

- expenses (list): Accumulator storing dictionaries with expense type and amount.

- expense\_type (str): User-entered expense type.

- amount (float): User-entered expense amount.

Logical Steps:

1. Initialize an empty list for expenses.

2. Prompt for expense type; exit loop if 'done' is entered.

3. If 'done' is entered with no expenses, prompt for at least one expense.

4. Prompt for amount and convert to float, catching ValueError for non-numeric inputs.

5. Validate amount is positive (>0).

6. Append expense dictionary to list.

7. Return the expenses list.

Returns: expenses (list): List of dictionaries containing expense type and amount.

2. Function Name: analyze\_expenses

Description: Uses reduce to calculate total expenses, highest expense, and lowest expense with their types.

Parameters:

- expenses (list): List of dictionaries with expense type and amount.

Variables:

- total (float): Total expenses calculated using reduce.

- highest (dict): Dictionary with type and amount of the highest expense.

- lowest (dict): Dictionary with type and amount of the lowest expense.

Logical Steps:

1. Use reduce to sum all expense amounts, starting with 0.0.

2. Use reduce to find the expense with the highest amount, starting with a dummy expense of negative infinity.

3. Use reduce to find the expense with the lowest amount, starting with a dummy expense of positive infinity.

4. Return total, highest, and lowest.

Returns: total (float): Sum of expenses; highest (dict): Highest expense details; lowest (dict): Lowest expense details.

3. Function Name: main

Description: Orchestrates expense collection, analysis, and result display.

Parameters: None

Variables:

- expenses (list): List of expense dictionaries from collect\_expenses.

- total (float): Total expenses from analyze\_expenses.

- highest (dict): Highest expense details from analyze\_expenses.

- lowest (dict): Lowest expense details from analyze\_expenses.

Logical Steps:

1. Call collect\_expenses to get the list of expenses.

2. Call analyze\_expenses to compute total, highest, and lowest expenses.

3. Display total expenses, highest expense with type, and lowest expense with type.

Returns: None

Logical Steps:

1. Call the main function.

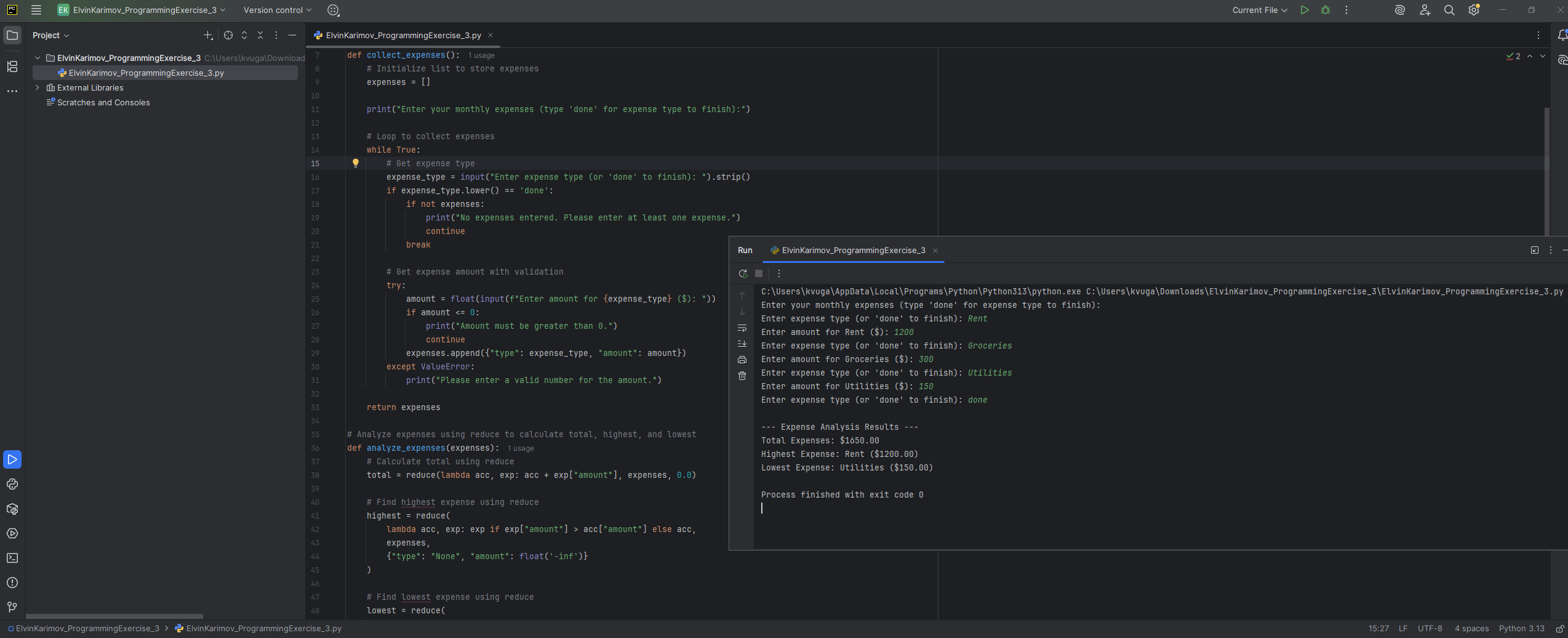
2. Within main:

a. Call collect\_expenses to collect user inputs.

b. Call analyze\_expenses to process the expenses.

c. Display the results.

Link to Repository: <https://github.com/SoldCarton86685/ElvinKarimov_ProgrammingExercise_3>

Output Screenshot:

Example output:

Enter your monthly expenses (type 'done' for expense type to finish):

Enter expense type (or 'done' to finish): Rent

Enter amount for Rent ($): 1200

Enter expense type (or 'done' to finish): Groceries

Enter amount for Groceries ($): 300

Enter expense type (or 'done' to finish): Utilities

Enter amount for Utilities ($): 150

Enter expense type (or 'done' to finish): done

--- Expense Analysis Results ---

Total Expenses: $1650.00

Highest Expense: Rent ($1200.00)

Lowest Expense: Utilities ($150.00)

Debugging Process:

The program was designed with robust input validation to prevent bugs. Using PyCharm’s debugger, I tested potential issues:

1. Set a breakpoint in collect\_expenses at the amount input line to inspect user inputs.

2. Ran in Debug mode with test inputs like non-numeric amounts (“abc”), negative amounts (-100), and empty submissions.

3. Verified that ValueError and positive amount checks caught invalid inputs, prompting the user again.

4. Tested analyze\_expenses with a breakpoint in the reduce functions to ensure correct accumulation and comparison.

5. Edge cases tested: single expense (highest/lowest are the same), identical amounts (first occurrence kept), and large inputs (no crashes).

6. No bugs were found, as validation ensured robust handling of all inputs and reduce correctly computed results.