**Name:** Jadyn Brabham  
**Date Created:** 02/10/2025

**Program Description:**This program will ask the user to input the type and amount of their monthly expenses. It will then analyze the expenses by calculating the total expense, finding the highest expense, and finding the lowest expense. The program uses the reduce function to process the list of expenses and display the results of the total expense, highest expense, and lowest expense to the user.

**Functions used in the Program (list in order as they are called):**1. **Function Name:** main()  
**Description:** The main function asks the user to input their monthly expenses, processes the input, and displays the total, highest, and lowest expense. It uses the total\_expense(), highest\_expense(), and lowest\_expense() functions to perform the calculations.

**Parameters:** None  
**Variables:**

1. expenses: a list that holds the type and amount of the user’s monthly expenses in tuples
2. expense\_type: a string holding the type of monthly expense the user inputs
3. amount: a float representing the expense amount
4. total: a float representing the sum of all the expenses
5. highest: a tuple representing the highest expense as (expense\_type, amount)
6. lowest: a tuple representing the lowest expense as (expense\_type, amount)

**Logical Steps:**

1. Define an empty list, expenses, to store the type and amount of each monthly expense input by the user.
2. Use a while loop to continually ask the user to input an expense type and amount.
3. Break the loop when the user types ‘done’.
4. If the user enters a valid amount, append the expense as a tuple to the expenses list.
5. After all the expenses are entered, call the total\_expense(), highest\_expense(), and lowest\_expense() functions to calculate the results.
6. Display the total, highest, and lowest expense.

**Returns:** The function does not return anything, instead it prints the output to the user.

2. **Function Name:** total\_expense(expenses)

**Description:** The total\_expense(expenses) function calculates the sum of all the expenses using the reduce() function to accumulate the total.  
**Parameters:**

1. expenses: a list of tuples, which contains the expense type and amount

**Variables:**

1. acc: accumulator variable that holds the running total of all expenses
2. expense: each individual expense tuple from the expenses list

**Logical Steps**:

1. Use the reduce() function to iterate over the expenses list.
2. For each expense, add the second element of the tuple (expense[1], amount) to the accumulator.
3. Return the total accumulated sum.

**Returns:**

1. total: a float representing the total accumulated sum

3.**Function Name:** highest\_expense(expenses)

**Description:** The highest\_expense(expenses) function finds the highest expense in list using the reduce() function to compare each expense’s amount.

**Parameters:**

1. expenses: a list of tuples, which contains the expense type and amount

**Variables:**

1. acc: accumulator variable that holds the current highest expense
2. expense: each individual expense tuple from the expenses list

**Logical Steps:**

1. Use the reduce() function to iterate over the expenses list.
2. Compare each expense’s amount (expense[1]) with the current highest expense (acc[1]).
3. If the current expense’s amount is greater, update the accumulator to the current expense.
4. Return the highest expense as a tuple

**Returns:**

1. highest: a tuple containing the type and amount of the highest expense.

4.**Function Name:** lowest\_expense(expenses)

**Description:** The lowest\_expense(expenses) function finds the lowest expense in the list using the reduce() function to compare each expense’s amount.

**Parameters:**

1. expenses: a list of tuples, which contains the expense type and amount

**Variables:**

1. acc: accumulator variable that holds the current lowest expense
2. expense: each individual expense tuple from the expenses list

**Logical Steps:**

1. Use the reduce() function to iterate over the expenses list.
2. Compare each expense’s amount (expense[1]) with the current lowest expense (acc[1]).
3. If the current expense’s amount is smaller, update the accumulator to the current expense.
4. Return the lowest expense as a tuple.

**Returns:**

1. lowest: a tuple containing the type and amount of the lowest expense

**Logical Steps:**

1. Call the main() function
2. The total\_expense(expenses), highest\_expense(expenses), and the lowest\_expense(expenses) function are called in the main() function.

**Link to your repository:** https://github.com/jbrabham23/COP2373/tree/master/Programming%202

**Screenshot of the program working:**

**A screenshot of a computer

AI-generated content may be incorrect.**