**Monthly Budget Calculation Algorithm**

1. **Step-by-Step Sequencing Algorithm**

**Step 1: Start**

Display a welcome message for the user.

**Step 2: Input Monthly Income**

Prompt the user:

Enter your total monthly income:

Store the values as "income".

**Step 3: Input Fixed Expenses**

Prompt the user for expense values:

Enter your rent amount: Store as "rent"

Enter your utility amount: Store as "utilities"

Calculate the total fixed expenses:

fixed\_expenses = rent + utilities

**Step 4: Number of Variable Expenses**

Prompt:

How many types of variable expenses do you have this month?

Store the value as num\_variable\_expenses

**Step 5: Input and Sum Variable Expenses (Using Iterative Loop)**

Initialize variable\_expenses = 0

For each expense (i from 1 to num\_variable\_expenses):

Prompt: Enter variable expense #i:

Add the input to variable\_expenses

**Step 6: Total Expenses and Budget Calculation**

total\_expenses = fixed\_expenses + variable\_expenses

remaining\_budget = income - total\_expenses

**Step 7: Conditional Selection – Expense Warning**

If remaining\_budget < 0:

Display: "Warning: Your expenses exceed your income!"

Else if remaining\_budget == 0:

Display: "You've balanced your budget perfectly!"

Else:

Display: "You have $[remaining\_budget] remaining after expenses."

**Step 8: End**

Display: "Thank you for using the Monthly Budget Calculator."

2. **Debugging Logical Errors in Budget Calculation**

Suppose the remaining budget is miscalculated under certain conditions (e.g., incorrect loop total, wrong variable used).

**Debugging Techniques**

* **Print Statements (Manual Debugging)**

Insert print statements at critical points:

* Print income, fixed, variable, and total expenses before the final calculation. This helps check whether each component is being computed as expected.
* **Variable Tracing**

Use a debugger tool (such as IDLE, PyCharm, or Visual Studio Code) to:

Step through each iteration of the loop.

Observe how variable values change, especially "variable\_expenses".

* **Unit Testing Small Components**

Test each part of the program separately:

Confirm that the summing of the variable expenses works correctly with known input.

Ensure the final subtraction (income-total\_expenses) uses correct, up-to-date values.

* **Edge Case Testing**

Run the algorithm with different scenarios:

No variable expenses (check 0 handling).

Expenses exceeding income.

Income equals total expenses.

This helps identify logic errors hidden during normal cases.

* **Code review or Peer Debugging**

Ask a peer to review the logic. Sometimes, another eye can spot the error faster.