PROJECT REPORT: MATRIX CALCULATOR USING PYTHON

---

1. Cover Page

Project Title: Matrix Calculator Using Python
Course / Subject: —Introduction To Problem Solving(CSE)
Submitted By: —Vivek Yadav

Registeration No.:-25BCE10796
Institution: —Vellore Institute of Technology Bhopal

---

2. Introduction

Matrix operations form the foundation of many scientific and engineering computations. This project focuses on developing an interactive command-line Matrix Calculator capable of performing essential matrix operations using Python and NumPy. The system allows users to input matrices and execute operations such as inverse, transpose, addition, subtraction, multiplication, and identity checking. The goal is to create a simple, modular, and educational tool.

---

3. Problem Statement

Students often struggle to manually compute matrix operations, especially for higher-order matrices. Existing tools may be complex or require installation of heavy software. The problem is to **design a lightweight, interactive, and user-friendly matrix calculator** capable of validating user inputs and performing essential matrix computations efficiently.

---

4. Functional Requirements

1. The system must allow users to input matrix dimensions and values.
2. The system must validate numeric entries and shape consistency.
3. The system must perform:
    - Matrix Inverse (if possible)
    - Matrix Transpose
    - Matrix Addition
    - Matrix Subtraction

    - Matrix Multiplication
    - Identical Matrix Check
4. The system must display errors for invalid operations.
5. The system must allow optional entry of a second matrix.

---

5. Non-functional Requirements

- **Usability:** The interface should be easy to navigate with clear prompts.
- **Reliability:** Must correctly validate inputs and avoid crashes.
- **Efficiency:** Matrix operations should be computed quickly using NumPy.
- **Maintainability:** Code should be modular and readable.
- **Portability:** Should run on any system with Python and NumPy installed.

---

6. System Architecture

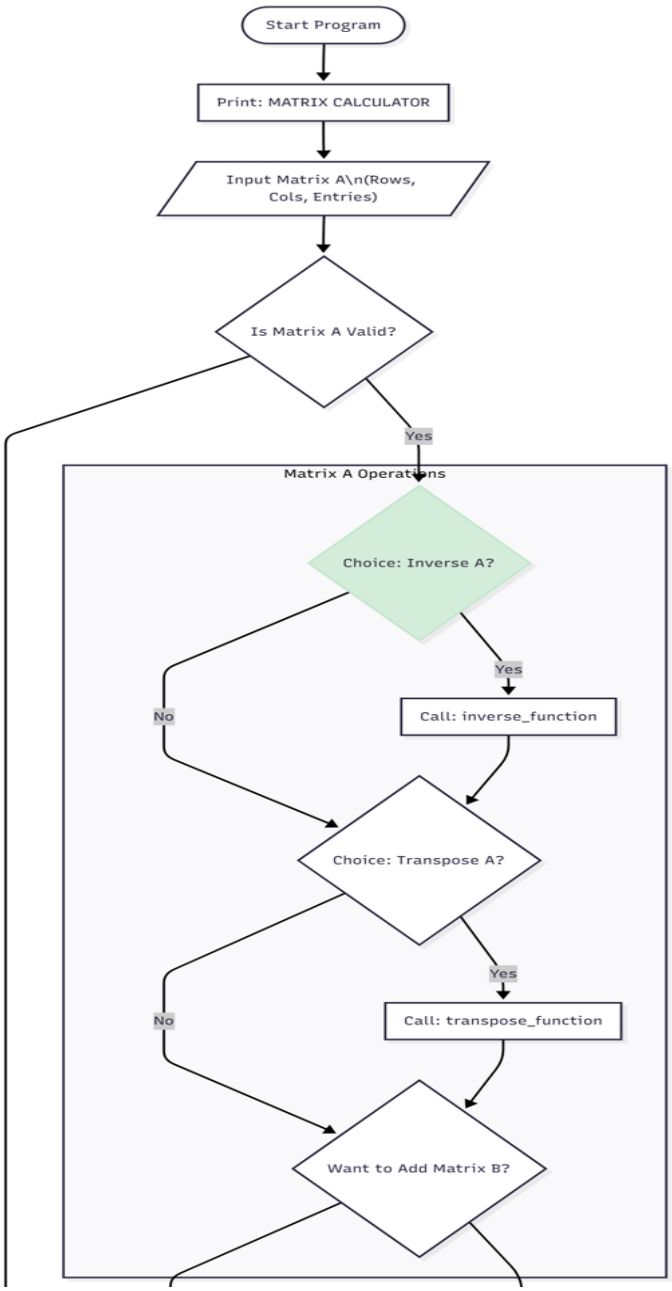
The system follows a **modular functional architecture**, structured as: - Input Validation Module - Matrix Operations Module - Identity Check Module - Controller (Main Program)

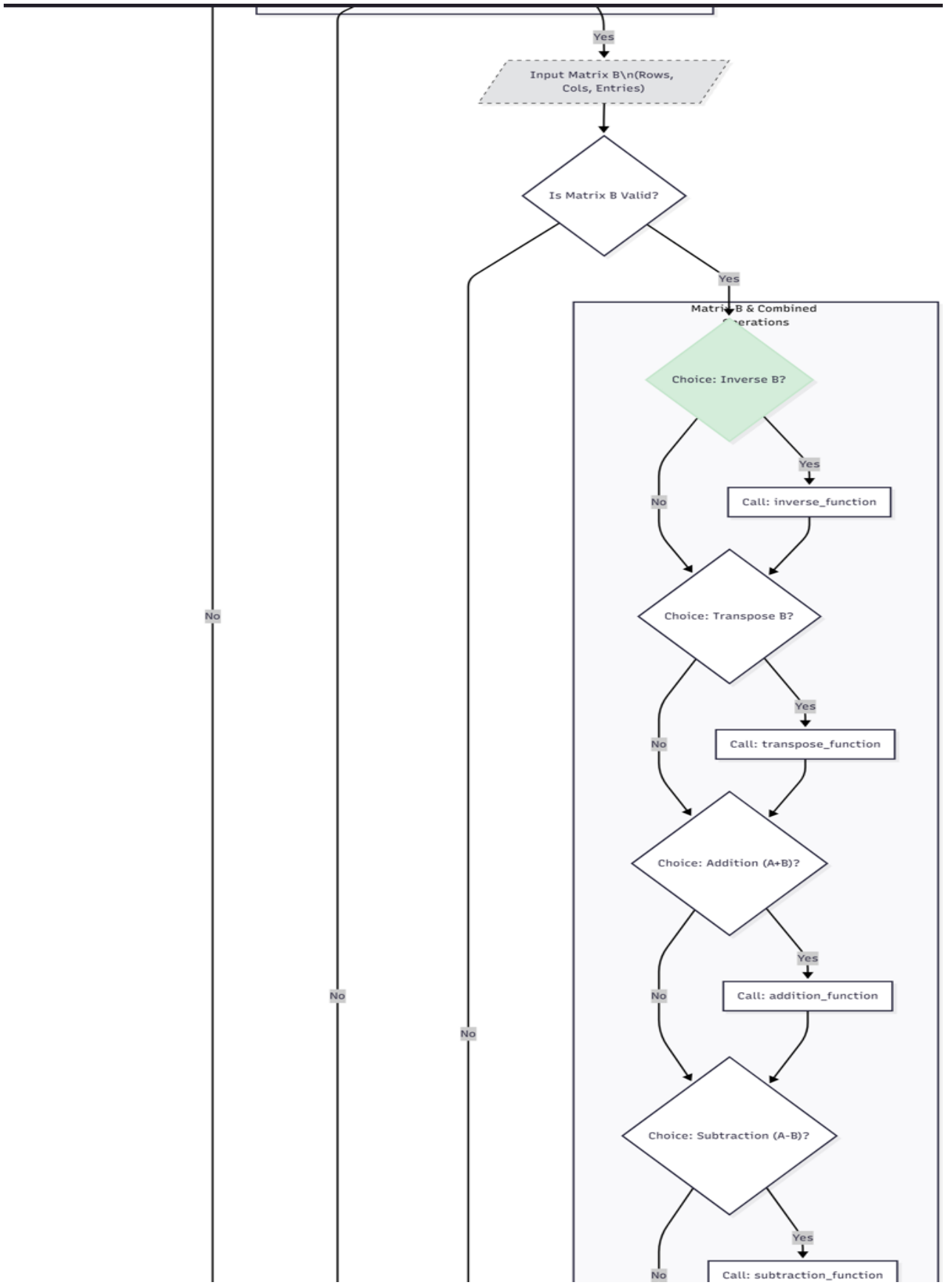
All components interact sequentially and are triggered based on user choices.

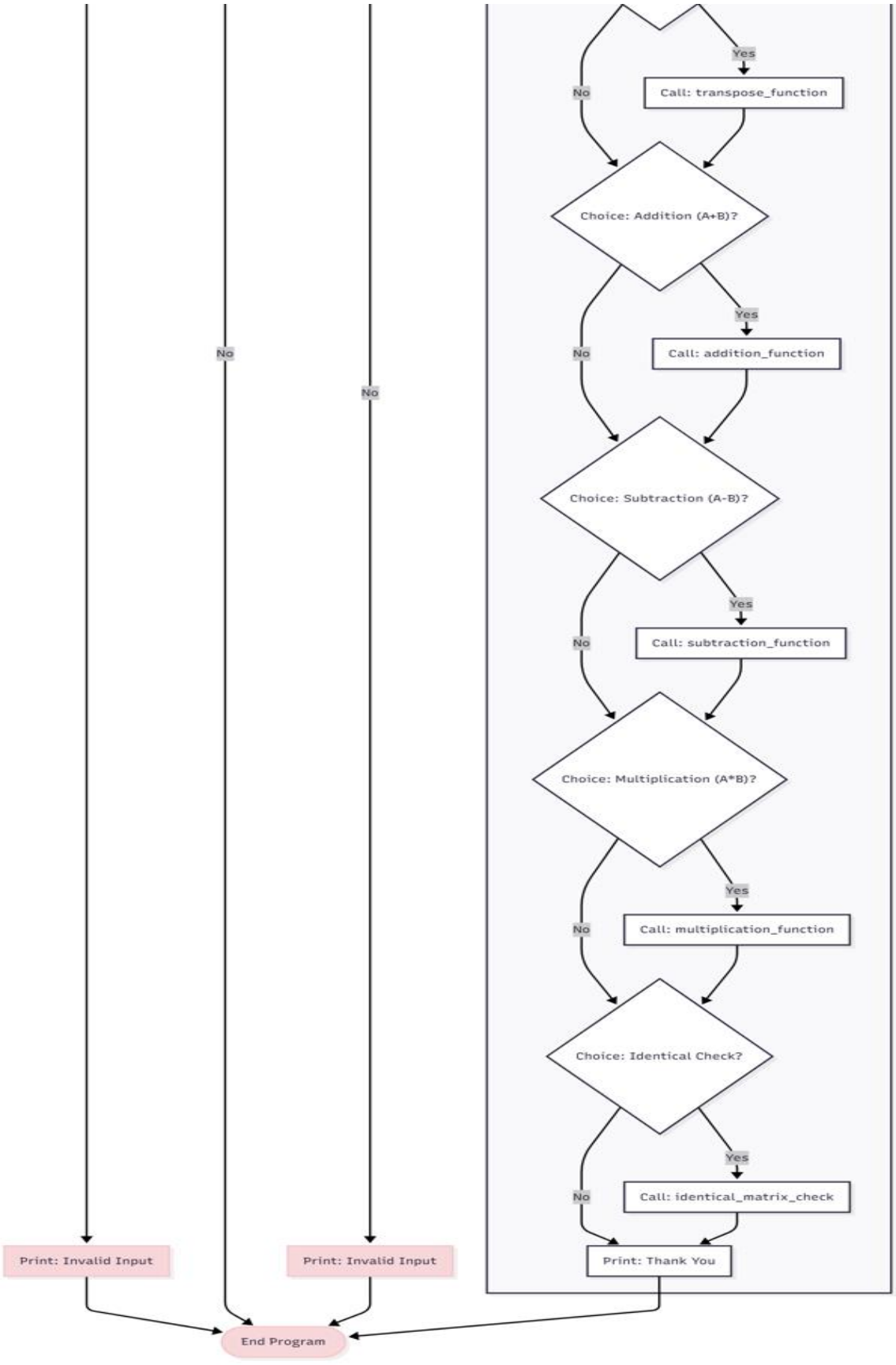
---

7. Design Diagrams

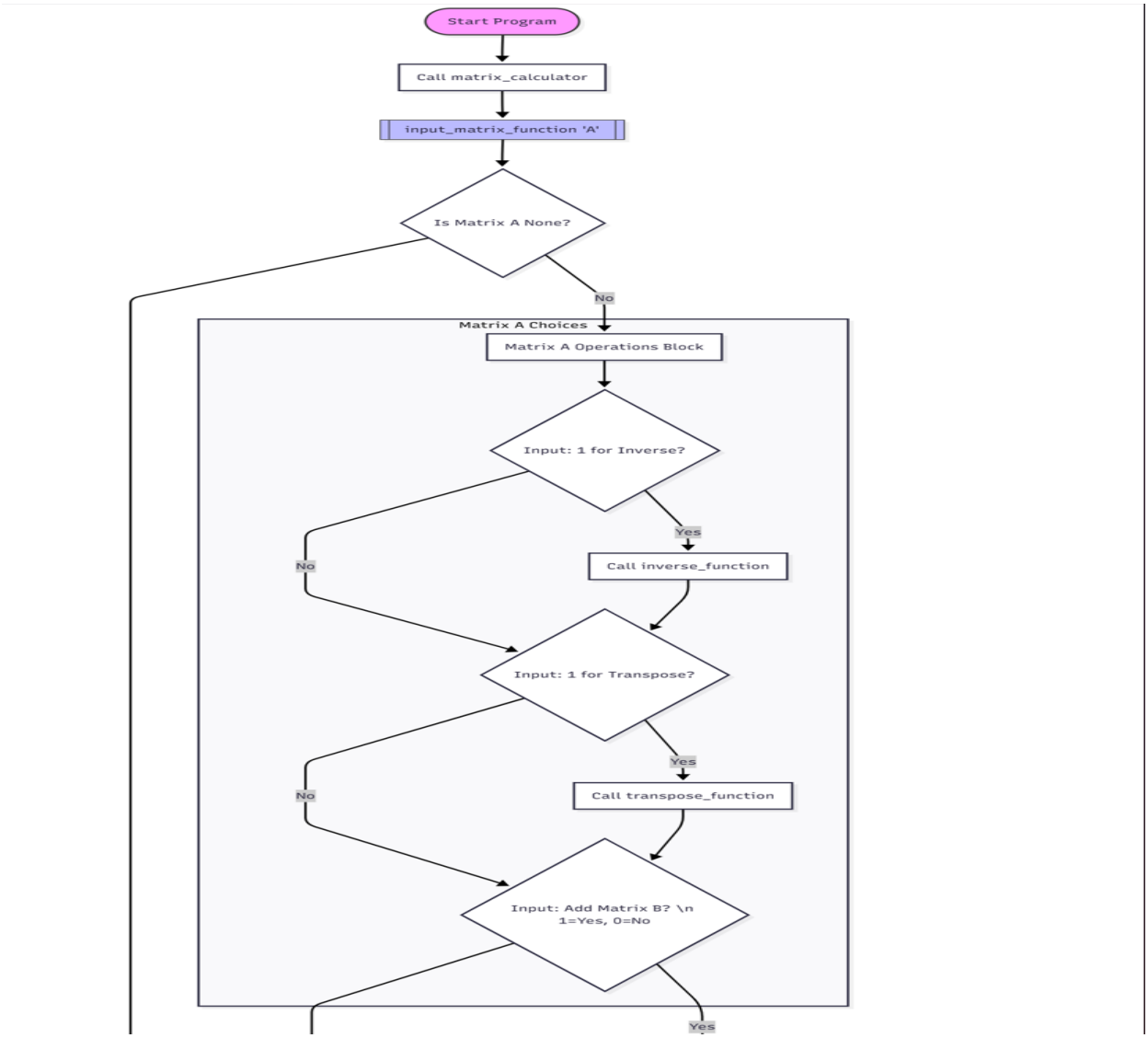
Workflow Diagram

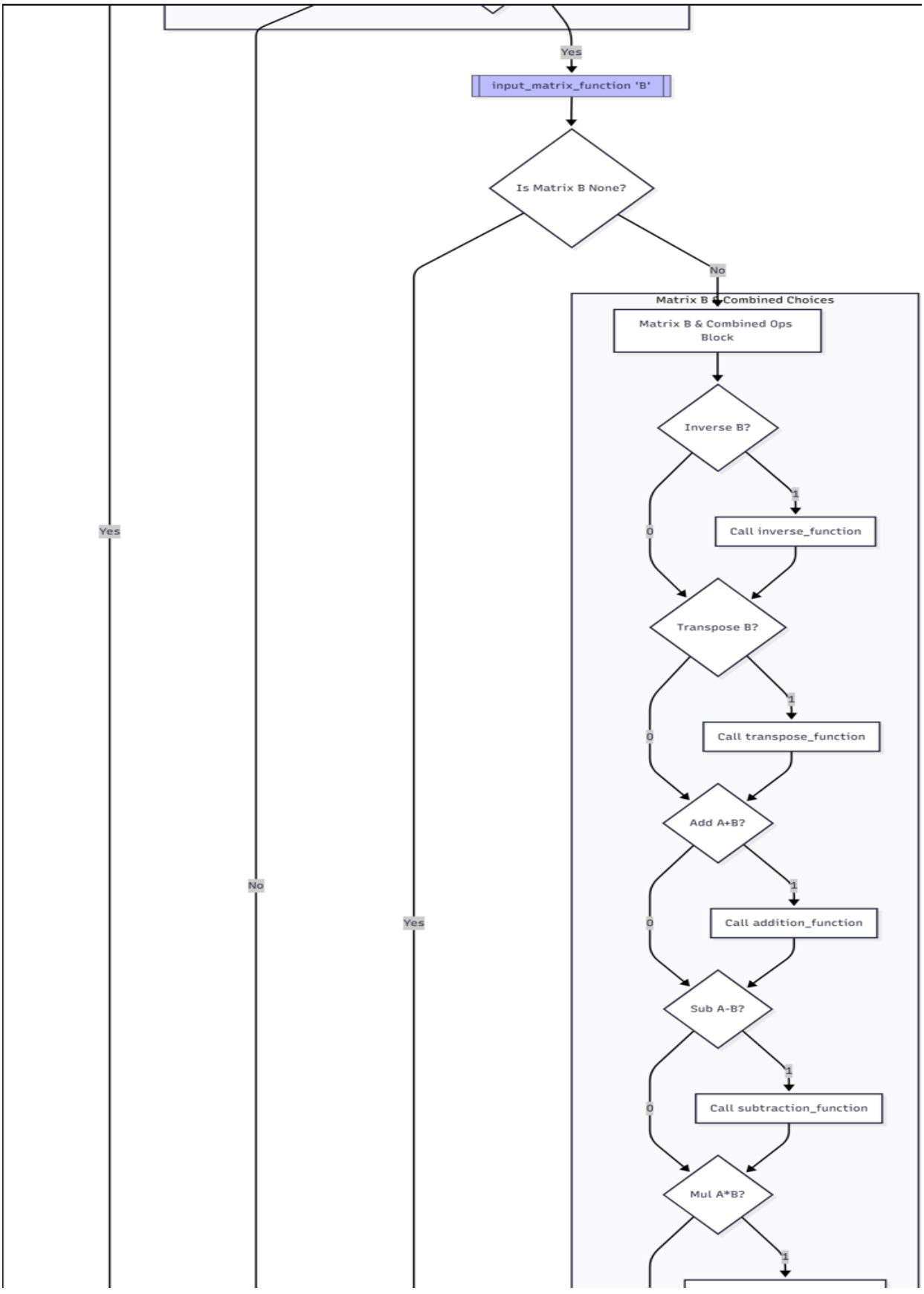

Start Program
Print: MATRIX CALCULATOR
Input Matrix A\n(Rows, Cols, Entries)
Is Matrix A Valid?
Yes
Matrix A Operations
Choice: Inverse A?
Yes
No
Call: inverse_function
Choice: Transpose A?
Yes
No
Call: transpose_function
Want to Add Matrix B?

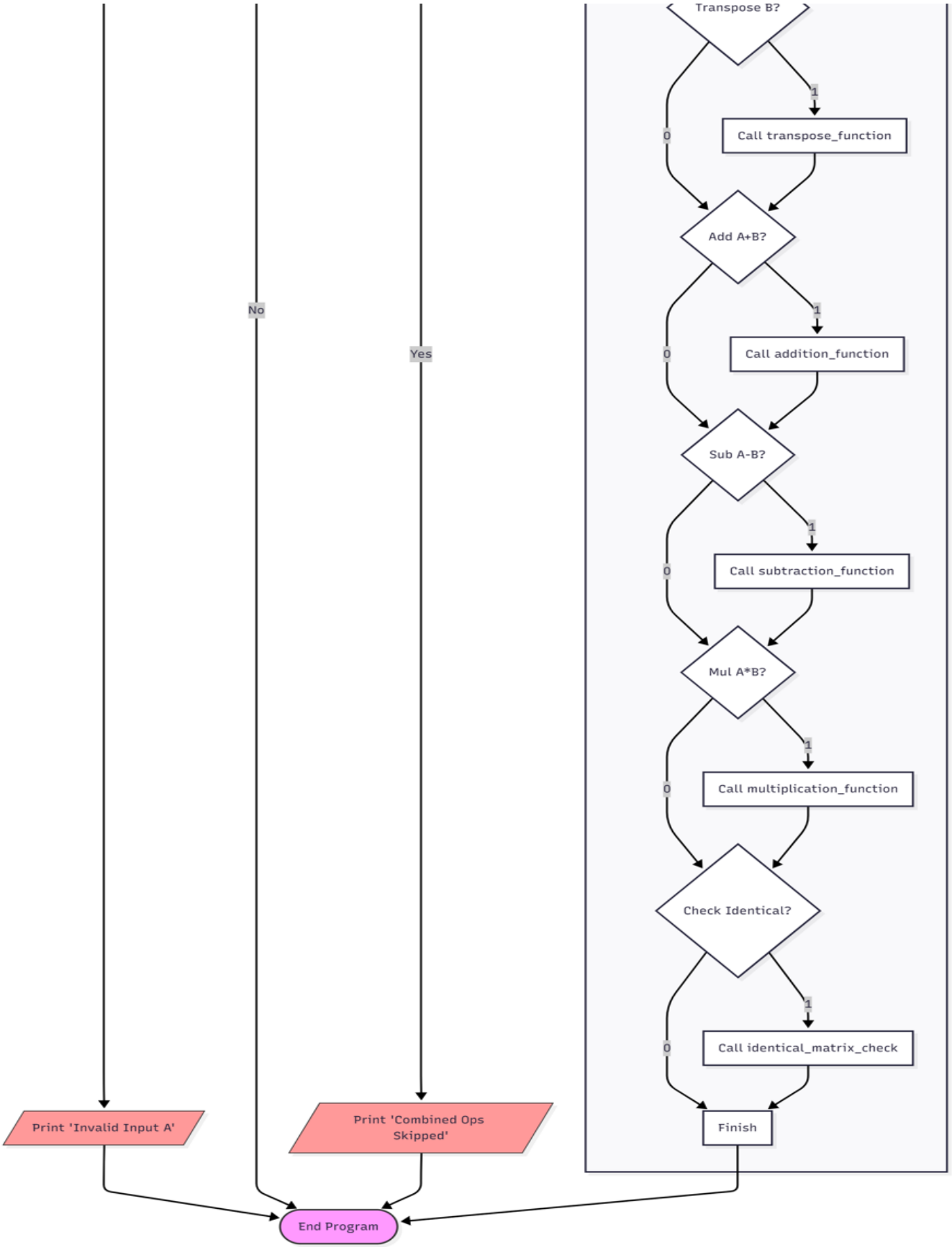

Yes
Input Matrix B\n(Rows, Cols, Entries)
Is Matrix B Valid?
Yes
Choice: Inverse B?
Yes
No
Call: inverse_function
Choice: Transpose B?
Yes
No
Call: transpose_function
Choice: Addition (A+B)?
Yes
No
Call: addition_function
Choice: Subtraction (A-B)?
Yes
No
Call: subtraction_function
No
No
No


Yes
No
Call: transpose_function
Choice: Addition (A+B)?
Yes
No
Call: addition_function
Choice: Subtraction (A-B)?
Yes
No
Call: subtraction_function
Choice: Multiplication (A*B)?
Yes
No
Call: multiplication_function
Choice: Identical Check?
Yes
No
Call: identical_matrix_check
Print: Thank You
No
No
Print: Invalid Input
Print: Invalid Input
End Program

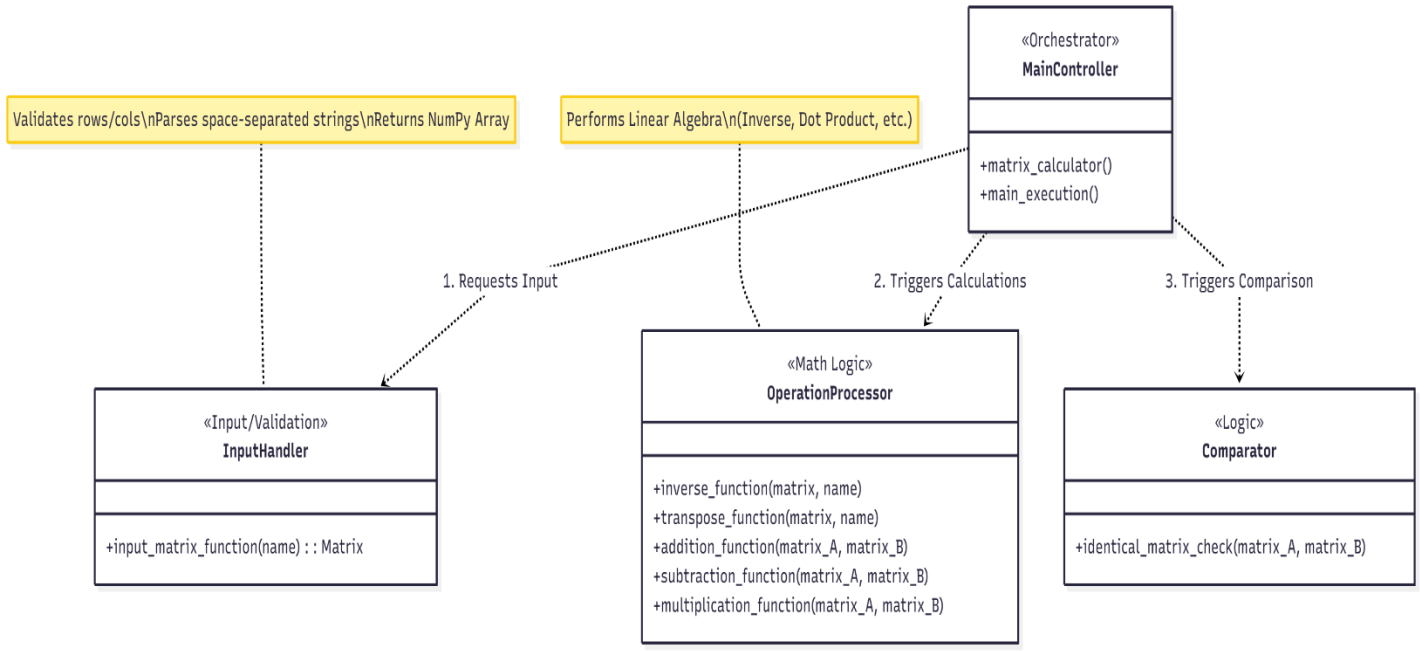
Sequence Diagram




Yes
input_matrix_function 'B'
Is Matrix B None?
No
Matrix B + Combined Choices
Matrix B & Combined Ops Block
Inverse B?
1
0
Call inverse_function
Transpose B?
1
0
Call transpose_function
Add A+B?
1
0
Call addition_function
Sub A-B?
1
0
Call subtraction_function
Mul A*B?
1
Yes
No
Yes


No
Yes
Transpose B?
1
Call transpose_function
0
Add A+B?
1
Call addition_function
0
Sub A-B?
1
Call subtraction_function
0
Mul A*B?
1
Call multiplication_function
0
Check Identical?
1
Call identical_matrix_check
0
Finish
Print 'Invalid Input A'
Print 'Combined Ops Skipped'
End Program

Class / Component Diagram



---

8. Design Decisions & Rationale

- **NumPy chosen** for efficient matrix computation.
- **Functional approach** selected to keep design simple and readable.
- **Validation loops** ensure robust user input handling.
- **Optional Matrix B** allows both single and dual matrix operations.

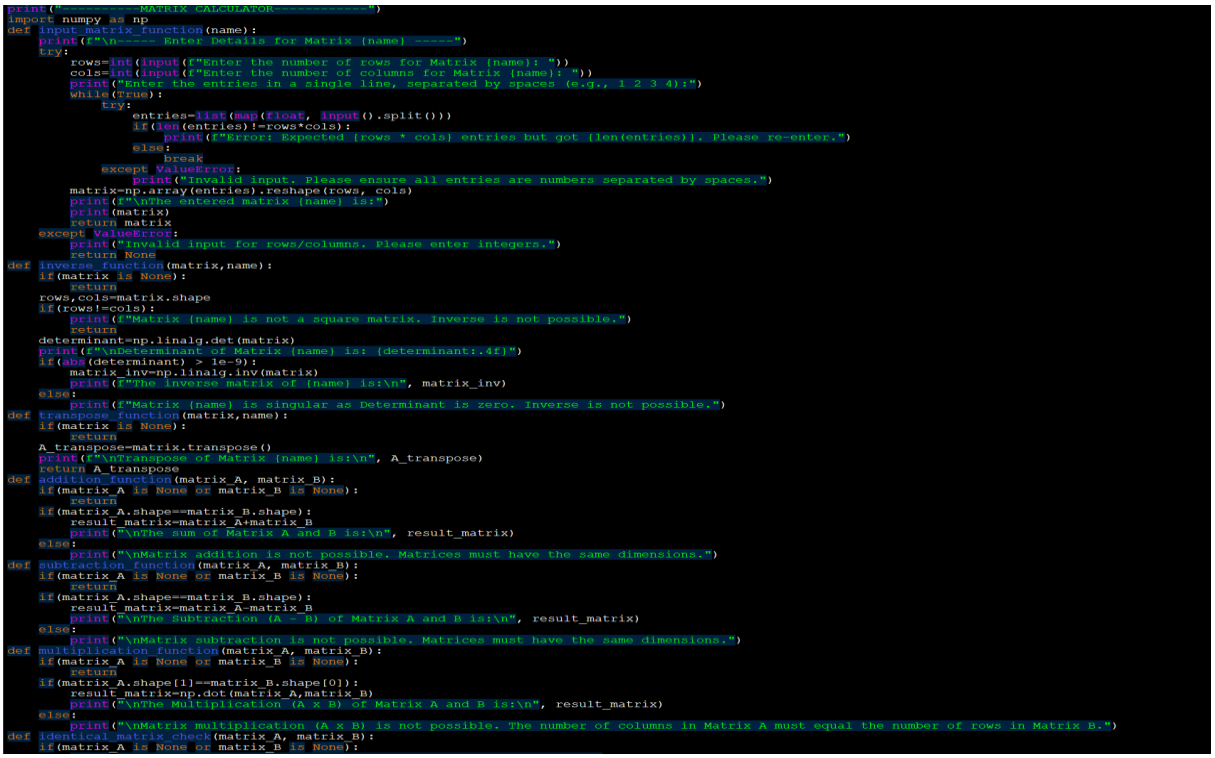
---

9. Implementation Details

Implementation is modular with functions such as: - `input_matrix_function()` — handles matrix input and validation. - `inverse_function()` — calculates matrix inverse with singularity check. - `transpose_function()` — returns matrix transpose. - `addition_function()` — adds two matrices. - `subtraction_function()` — subtracts matrices. - `multiplication_function()` — multiplies matrices. - `identical_matrix_check()` — checks if matrices are identical. - `matrix_calculator()` — main driver controlling program flow.

Programming Language: **Python**
Library Used: **NumPy**

10. Screenshots / Result



```
print("----------MATRIX CALCULATOR----------")
import numpy as np
def input_matrix_function(name):
    print(f"\n----- Enter Details for Matrix {name} -----")
    try:
        rows=int(input(f"Enter the number of rows for Matrix {name}: "))
        cols=int(input(f"Enter the number of columns for Matrix {name}: "))
        print("Enter the entries in a single line, separated by spaces (e.g., 1 2 3 4):")
        while(True):
            try:
                entries=list(map(float, input().split()))
                if(len(entries)!=rows*cols):
                    print(f"Error: Expected {rows * cols} entries but got {len(entries)}. Please re-enter.")
                else:
                    break
            except ValueError:
                print("Invalid input. Please ensure all entries are numbers separated by spaces.")
        matrix=np.array(entries).reshape(rows, cols)
        print(f"\nThe entered matrix {name} is:")
        print(matrix)
        return matrix
    except ValueError:
        print("Invalid input for rows/columns. Please enter integers.")
        return None
def inverse_function(matrix,name):
    if(matrix is None):
        return
    rows,cols=matrix.shape
    if(rows!=cols):
        print(f"Matrix {name} is not a square matrix. Inverse is not possible.")
        return
    determinant=np.linalg.det(matrix)
    print(f"\nDeterminant of Matrix {name} is: {determinant:.4f}")
    if(abs(determinant) > 1e-9):
        matrix_inv=np.linalg.inv(matrix)
        print(f"The inverse matrix of {name} is:\n", matrix_inv)
    else:
        print(f"Matrix {name} is singular as Determinant is zero. Inverse is not possible.")
def transpose_function(matrix,name):
    if(matrix is None):
        return
    A_transpose=matrix.transpose()
    print(f"\nTranspose of Matrix {name} is:\n", A_transpose)
    return A_transpose
def addition_function(matrix_A, matrix_B):
    if(matrix_A is None or matrix_B is None):
        return
    if(matrix_A.shape==matrix_B.shape):
        result_matrix=matrix_A+matrix_B
        print("\nThe sum of Matrix A and B is:\n", result_matrix)
    else:
        print("\nMatrix addition is not possible. Matrices must have the same dimensions.")
def subtraction_function(matrix_A, matrix_B):
    if(matrix_A is None or matrix_B is None):
        return
    if(matrix_A.shape==matrix_B.shape):
        result_matrix=matrix_A-matrix_B
        print("\nThe Subtraction (A - B) of Matrix A and B is:\n", result_matrix)
    else:
        print("\nMatrix subtraction is not possible. Matrices must have the same dimensions.")
def multiplication_function(matrix_A, matrix_B):
    if(matrix_A is None or matrix_B is None):
        return
    if(matrix_A.shape[1]==matrix_B.shape[0]):
        result_matrix=np.dot(matrix_A,matrix_B)
        print("\nThe Multiplication (A x B) of Matrix A and B is:\n", result_matrix)
    else:
        print("\nMatrix multiplication (A x B) is not possible. The number of columns in Matrix A must equal the number of rows in Matrix B.")
def identical_matrix_check(matrix_A, matrix_B):
    if(matrix_A is None or matrix_B is None):
```

```
    if(matrix_A is None or matrix_B is None):
        return
    if(matrix_A.shape==matrix_B.shape):
        result_matrix=matrix_A-matrix_B
        print("\nThe Subtraction (A - B) of Matrix A and B is:\n", result_matrix)
    else:
        print("\nMatrix subtraction is not possible. Matrices must have the same dimensions.")
def multiplication_function(matrix_A, matrix_B):
    if(matrix_A is None or matrix_B is None):
        return
    if(matrix_A.shape[1]==matrix_B.shape[0]):
        result_matrix=np.dot(matrix_A,matrix_B)
        print("\nThe Multiplication (A x B) of Matrix A and B is:\n", result_matrix)
    else:
        print("\nMatrix multiplication (A x B) is not possible. The number of columns in Matrix A must equal the number of rows in Matrix B.")
def identical_matrix_check(matrix_A, matrix_B):
    if(matrix_A is None or matrix_B is None):
        print("Cannot check for equality: One or both matrices are invalid.")
        return
    if(np.array_equal(matrix_A, matrix_B)):
        print("\n The matrices are Identical.")
    else:
        if(matrix_A.shape!=matrix_B.shape):
            print(f"\n The matrices are Not Identical because they have different dimensions: {matrix_A.shape} vs {matrix_B.shape}.")
        else:
            print("\n The matrices are Not Identical (same shape, but different elements).")
def matrix_calculator():
    matrix_A=input_matrix_function("A")
    if(matrix_A is None):
        print("\nProgram terminated due to invalid input for Matrix A.")
        print("\n--------Thank You for using Matrix Calculator----------")
        return
    print("\n----- Operations for Matrix A -----")
    try:
        if int(input("ENTER (1) IF YOU WANT INVERSE OF MATRIX A OTHERWISE (0):"))==1:
            inverse_function(matrix_A, "A")
        if int(input("ENTER (1) IF YOU WANT TRANSPOSE OF MATRIX A OTHERWISE (0):"))==1:
            transpose_function(matrix_A, "A")
    except ValueError:
        print(" Invalid input for choice. Skipping remaining Matrix A operations.")
    try:
        add_matrix_B_choice = int(input("\nIf you want to add another matrix (B) for combined operations (1), otherwise (0): "))
    except ValueError:
        print("Invalid input for choice. Skipping combined operations.")
        add_matrix_B_choice=0
    if (add_matrix_B_choice==1):
        matrix_B=input_matrix_function("B")
        if (matrix_B is None):
            print("\nCombined operations skipped due to invalid input for Matrix B.")
        else:
            print("\n----- Operations for Matrix B and Combined Operations (A & B) -----")
            try:
                if int(input("ENTER (1) IF YOU WANT INVERSE OF MATRIX B OTHERWISE (0):"))==1:
                    inverse_function(matrix_B, "B")
                if int(input("ENTER (1) IF YOU WANT TRANSPOSE OF MATRIX B OTHERWISE (0):"))==1:
                    transpose_function(matrix_B, "B")
                if int(input("ENTER (1) IF YOU WANT Addition of Matrix A and B OTHERWISE (0):"))==1:
                    addition_function(matrix_A, matrix_B)
                if int(input("ENTER (1) If YOU WANT Subtraction of Matrix A and B (A-B) OTHERWISE (0):"))==1:
                    subtraction_function(matrix_A, matrix_B)
                if int(input("ENTER (1) IF YOU WANT Multiplication of Matrix A and B (A*B) OTHERWISE (0):"))==1:
                    multiplication_function(matrix_A, matrix_B)
                if int(input("ENTER (1) IF YOU WANT TO CHECK WHETHER Matrix A and Matrix b are Identical OTHERWISE (0):"))==1:
                    identical_matrix_check(matrix_A,matrix_B)
            except ValueError:
                print(" Invalid input for choice. Skipping remaining combined operations.")
    print("\n--------Thank You for using Matrix Calculator----------")
if __name__=="__main__":
    matrix_calculator()
```

```
----------MATRIX CALCULATOR------------

----- Enter Details for Matrix A -----
Enter the number of rows for Matrix A: 2
Enter the number of columns for Matrix A: 2
Enter the entries in a single line, separated by spaces (e.g., 1 2 3 4):
1 2 3 4

The entered matrix A is:
[[1. 2.]
 [3. 4.]]

----- Operations for Matrix A -----
ENTER (1) IF YOU WANT INVERSE OF MATRIX A OTHERWISE (0):1

Determinant of Matrix A is: -2.0000
The inverse matrix of A is:
 [[-2.   1. ]
 [ 1.5 -0.5]]
ENTER (1) IF YOU WANT TRANSPOSE OF MATRIX A OTHERWISE (0):1

Transpose of Matrix A is:
 [[1. 3.]
 [2. 4.]]

If you want to add another matrix (B) for combined operations (1), otherwise (0): 1

----- Enter Details for Matrix B -----
Enter the number of rows for Matrix B: 2
Enter the number of columns for Matrix B: 2
Enter the entries in a single line, separated by spaces (e.g., 1 2 3 4):
4 5 6 7

The entered matrix B is:
[[4. 5.]
 [6. 7.]]

----- Operations for Matrix B and Combined Operations (A & B) -----
ENTER (1) IF YOU WANT INVERSE OF MATRIX B OTHERWISE (0):1

Determinant of Matrix B is: -2.0000
The inverse matrix of B is:
 [[-3.5  2.5]
 [ 3.  -2. ]]
ENTER (1) IF YOU WANT TRANSPOSE OF MATRIX B OTHERWISE (0):1

Transpose of Matrix B is:
 [[4. 6.]
 [5. 7.]]
ENTER (1) IF YOU WANT Addition of Matrix A and B OTHERWISE (0):1

The sum of Matrix A and B is:
 [[ 5.  7.]
 [ 9. 11.]]
ENTER (1) If YOU WANT Subtraction of Matrix A and B (A-B) OTHERWISE (0):1

The Subtraction (A - B) of Matrix A and B is:
 [[-3. -3.]
 [-3. -3.]]
ENTER (1) IF YOU WANT Multiplication of Matrix A and B (A*B) OTHERWISE (0):1

The Multiplication (A x B) of Matrix A and B is:
 [[16. 19.]
 [36. 43.]]
ENTER (1) IF YOU WANT TO CHECK WHETHER Matrix A and Matrix b are Identical OTHERWISE (0):1

 The matrices are Not Identical (same shape, but different elements).

--------Thank You for using Matrix Calculator----------
```

11. Testing Approach

Testing was performed using: - **Valid matrices** (square & non-square) - **Invalid inputs** (wrong entries, mismatched dimensions) - **Singular matrices** (determinant = 0) - **Large matrices** for performance validation - **Identical and non-identical matrices** for equality checks

Testing ensures correctness, stability, and input robustness.

---

12. Challenges Faced

- Handling invalid user inputs gracefully
- Managing dimension mismatches in operations
- Ensuring the program does not crash during unexpected input
- Maintaining readability while implementing multiple functions

---

13. Learnings & Key Takeaways

- Improved understanding of matrix operations using NumPy
- Experience with functional program design
- Exposure to error handling and input validation
- Importance of modular design for maintainability

---

14. Future Enhancements

- Adding a Graphical User Interface (GUI) using Tkinter or PyQt
- Adding support for matrix rank, eigenvalues, determinants-only mode
- Ability to save matrices to files
- Allowing fractional/complex numbers
- Storing previous operations in history

---

15. References

- NumPy Documentation: https://numpy.org/doc/
- Python Official Documentation: https://docs.python.org/
- Linear Algebra Resources
