Create a Python application using the NumPy library that allows users to input matrices and perform various matrix operations including addition, subtraction, multiplication, transpose, and determinant calculation. The application should include an interactive interface to display results in a structured format.

Project Title: Python Matrix Operations Tool Using NumPy

1. Introduction

This project involves creating a Python application that allows users to input matrices and perform various matrix operations such as addition, subtraction, multiplication, transpose, and determinant calculation. The application features an interactive command-line interface for ease of use and displays results in a structured format. The NumPy library is used for efficient numerical computation.

---

2. Functional Requirements

- Accept user input for two matrices, including their dimensions and elements.
- Validate input to ensure matrices have valid dimensions and correct number of elements.
- Provide options to perform operations:
    - Addition (A + B)
    - Subtraction (A - B)
    - Multiplication (A * B)
    - Transpose of A or B
    - Determinant of A or B (only if the matrix is square)
- Display the results clearly and formatted.
- Offer an exit option to quit the application.

---

3. Technology Stack

- Python 3.x
- NumPy library for matrix operations and numerical processing
- Command-line interface for interaction

---

4. Application Design and Implementation

Core Modules and Functions

- **get_matrix_input(matrix_name)**:
    - Prompts user to enter matrix dimensions and elements.
    - Validates and returns a NumPy array representing the matrix.
- **display_matrix(matrix, name)**:
    - Nicely formats and prints the matrix with a title.
- **matrix_operations_tool()**:
    - Main loop presenting options and handling user choices.
    - Performs matrix operations using NumPy.
    - Handles dimension checks and errors gracefully.

Code Snippet of Core Application

```
import numpy as np



def get_matrix_input(matrix_name):

  print(f"\nEnter dimensions for {matrix_name} (rows columns):")

      try:

      rows, cols = map(int, input().split())

      if rows <= 0 or cols <= 0:

      raise ValueError("Dimensions must be positive integers.")


    print(f"Enter {rows}x{cols} elements for {matrix_name} (row-wise, space-separated):")

      elements = []

      for i in range(rows):

      row = list(map(float, input(f"Row {i+1}: ").split()))

      if len(row) != cols:

            raise ValueError(f"Expected {cols} elements in row {i+1}, got {len(row)}.")

      elements.append(row)
```

```
            return np.array(elements)

            except ValueError as e:

        print(f"Error: {e}")

            return None


def display_matrix(matrix, name):

    print(f"\n{name}:")

            for row in matrix:

        print("[" + " ".join(f"{x:8.2f}" for x in row) + "]")


def matrix_operations_tool():

            print("=== Matrix Operations Tool ===")


            A = get_matrix_input("Matrix A")

            if A is None:

        print("Invalid input for Matrix A. Exiting.")

            return


            B = get_matrix_input("Matrix B")

            if B is None:

        print("Invalid input for Matrix B. Exiting.")

            return


            display_matrix(A, "Matrix A")
    display_matrix(B, "Matrix B")


    while True:

print("\nSelect an operation:")

    print("1. Addition (A + B)")

    print("2. Subtraction (A - B)")

    print("3. Multiplication (A * B)")

    print("4. Transpose of Matrix A")

    print("5. Transpose of Matrix B")

    print("6. Determinant of Matrix A")

    print("7. Determinant of Matrix B")

    print("8. Exit")


    choice = input("Enter choice (1-8): ")


    try:

    if choice == "1":

                if A.shape != B.shape:

            print("Error: Matrices must have the same dimensions for addition.")

                else:

            result = A + B

            display_matrix(result, "A + B")


    elif choice == "2":
```

```
                if A.shape != B.shape:

              print("Error: Matrices must have the same dimensions for subtraction.")

                else:

              result = A - B

              display_matrix(result, "A - B")


        elif choice == "3":

                if A.shape[1] != B.shape[0]:

              print("Error: Number of columns in A must equal number of rows in B for
multiplication.")

                else:

              result = np.dot(A, B)

              display_matrix(result, "A * B")


        elif choice == "4":

                result = A.T

           display_matrix(result, "Transpose of A")


        elif choice == "5":

                result = B.T

           display_matrix(result, "Transpose of B")


        elif choice == "6":

                if A.shape[0] != A.shape[1]:

              print("Error: Matrix A must be square for determinant calculation.")
```

```
                        else:

                    result = np.linalg.det(A)

                    print(f"\nDeterminant of A: {result:.2f}")


            elif choice == "7":

                        if B.shape[0] != B.shape[1]:

                    print("Error: Matrix B must be square for determinant calculation.")

                        else:

                    result = np.linalg.det(B)

                    print(f"\nDeterminant of B: {result:.2f}")


            elif choice == "8":

                print("Exiting Matrix Operations Tool.")

                        break


            else:

                print("Invalid choice. Please select 1-8.")


            except Exception as e:

            print(f"Error during operation: {e}")


if __name__ == "__main__":

    matrix_operations_tool()
```

5. Conclusion

This project demonstrates a user-interactive Python application for fundamental matrix operations using NumPy. It provides users with flexibility to input matrices and explores basic linear algebra concepts interactively, serving both educational and practical computation purposes.