Tina Borundia

1.Write a python code for element wise addition, subtraction, multiplication and division of two 4x4 matrices.

In [ ]:
```
!pip install numpy
```

In [31]:
```
import numpy
```

In [11]:
```
import numpy as np
matrix1 = np.array([[1, 2, 3, 4],
                    [5, 6, 7, 8],
                    [9, 10, 11, 12],
                    [13, 14, 15, 16]])

matrix2 = np.array([[4, 3, 2, 1],
                    [8, 7, 6, 5],
                    [12, 11, 10, 9],
                    [16, 15, 14, 13]])

addition_result = matrix1 + matrix2

subtraction_result = matrix1 - matrix2

multiplication_result = matrix1 * matrix2

epsilon = 1e-9
division_result = matrix1 / (matrix2 + epsilon)

print("Matrix 1:")
print(matrix1)
print("\nMatrix 2:")
print(matrix2)
print("\nElement-wise Addition:")
print(addition_result)
print("\nElement-wise Subtraction:")
print(subtraction_result)
print("\nElement-wise Multiplication:")
print(multiplication_result)
print("\nElement-wise Division:")
print(division_result)
```

```
Matrix 1:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]

Matrix 2:
[[ 4  3  2  1]
 [ 8  7  6  5]
 [12 11 10  9]
 [16 15 14 13]]

Element-wise Addition:
[[ 5  5  5  5]
 [13 13 13 13]
 [21 21 21 21]
 [29 29 29 29]]

Element-wise Subtraction:
[[-3 -1  1  3]
 [-3 -1  1  3]
 [-3 -1  1  3]
 [-3 -1  1  3]]

Element-wise Multiplication:
[[  4   6   6   4]
 [ 40  42  42  40]
 [108 110 110 108]
 [208 210 210 208]]

Element-wise Division:
[[0.25       0.66666667 1.5        4.        ]
 [0.625      0.85714286 1.16666667 1.6       ]
 [0.75       0.90909091 1.1        1.33333333]
 [0.8125     0.93333333 1.07142857 1.23076923]]
```

2. Write a program to sort a given array in descending order. You have to sort an entire array, in row wise and in column wise. Also compute the following: Mean, median, mode, variance, and standard deviation. (hint: Use Scipy for mode)

In [12]:
```
import numpy as np
from scipy import stats

array = np.array([[5, 3, 2, 7],
                  [8, 1, 4, 6],
                  [9, 11, 10, 12]])

sorted_array_row = np.sort(array, axis=1)[:, ::-1]

sorted_array_col = np.sort(array, axis=0)[::-1, :]

mean = np.mean(array)
median = np.median(array)
mode = stats.mode(array, axis=None)
variance = np.var(array)
std_dev = np.std(array)

print("Original Array:")
print(array)
print("\nSorted Row-wise in Descending Order:")
print(sorted_array_row)
print("\nSorted Column-wise in Descending Order:")
print(sorted_array_col)
print("\nMean:", mean)
print("Median:", median)
print("Mode:", mode.mode[0])
print("Variance:", variance)
print("Standard Deviation:", std_dev)
```

```
Original Array:
[[ 5  3  2  7]
 [ 8  1  4  6]
 [ 9 11 10 12]]

Sorted Row-wise in Descending Order:
[[ 7  5  3  2]
 [ 8  6  4  1]
 [12 11 10  9]]

Sorted Column-wise in Descending Order:
[[ 9 11 10 12]
 [ 8  3  4  7]
 [ 5  1  2  6]]

Mean: 6.5
Median: 6.5
Mode: 1
Variance: 11.916666666666666
Standard Deviation: 3.452052529534663
```

3.Randomly generate the marks of the 100 students in the range of 30 to 95. Write a NumPy program to compute the 60 percentiles for all elements in a given array. (Hint: use np.random.randint (start, stop, no_of_items) for list generation) (Hint: Use np.percentile)

In [14]:
```
import numpy as np

random_marks = np.random.randint(30, 96, 100)

percentile_60 = np.percentile(random_marks, 60)

print("Random Marks for 100 Students:")
print(random_marks)
print("\n60th Percentile:", percentile_60)
```

```
Random Marks for 100 Students:
[88 73 66 91 73 49 47 58 53 57 80 32 38 82 73 92 67 30 59 36 71 89 63 75
 50 60 37 83 42 59 77 44 84 57 85 48 75 56 57 79 31 60 73 79 68 57 63 39
 74 30 91 37 37 89 38 57 73 58 35 66 88 88 86 82 38 74 76 65 33 38 66 30
 74 43 84 79 72 50 40 90 33 77 41 86 38 45 87 37 82 49 68 67 74 93 63 79
 58 69 36 48]

60th Percentile: 71.4
```

4.You are given a space separated list of nine integers. Convert this list into a 3x3 NumPy array.

Eg: input: 1 2 3 4 5 6 7 8 9

Output: [[1 2 3][4 5 6][7 8 9]]

Further, compute the determinant and inverse of the array. Also compute the eigenvalues and eigenvectors of a given array.

In [ ]:
```
import numpy as np

input_str = input("Enter nine integers separated by spaces: ")
input_list = list(map(int, input_str.split()))

if len(input_list) != 9:
    print("Input should contain exactly nine integers.")
else:
    array_3x3 = np.array(input_list).reshape(3, 3)

    determinant = np.linalg.det(array_3x3)

    inverse = np.linalg.inv(array_3x3)

    eigenvalues, eigenvectors = np.linalg.eig(array_3x3)

    print("3x3 Array:")
    print(array_3x3)
    print("\nDeterminant:", determinant)
    print("\nInverse:")
    print(inverse)
    print("\nEigenvalues:")
    print(eigenvalues)
    print("\nEigenvectors:")
    print(eigenvectors)
```

5. Consider two matrices

M1=([[2,3,4], [6,5,2], [6,7,3]])

M2=([[1,4,2], [4,3,6],[5,9,8]])

Develop the python program for the following:

(1) matrix multiplication (dot product) (2) inner product (3) cross product (4) outer product

In [16]:
```
import numpy as np

M1 = np.array([[2, 3, 4], [6, 5, 2], [6, 7, 3]])
M2 = np.array([[1, 4, 2], [4, 3, 6], [5, 9, 8]])

matrix_multiplication = np.dot(M1, M2)

inner_product = np.sum(M1 * M2)

cross_product = np.sum(M1 - M2)

outer_product = M1 * M2

print("Matrix M1:")
print(M1)
print("\nMatrix M2:")
print(M2)
print("\nMatrix Multiplication (Dot Product):")
print(matrix_multiplication)
print("\nInner Product (Element-wise multiplication and sum):")
print(inner_product)
print("\nCross Product (Element-wise subtraction and sum):")
print(cross_product)
print("\nOuter Product (Element-wise multiplication):")
print(outer_product)
```

```
Matrix M1:
[[2 3 4]
 [6 5 2]
 [6 7 3]]

Matrix M2:
[[1 4 2]
 [4 3 6]
 [5 9 8]]

Matrix Multiplication (Dot Product):
[[34 53 54]
 [36 57 58]
 [49 72 78]]

Inner Product (Element-wise multiplication and sum):
190

Cross Product (Element-wise subtraction and sum):
-4

Outer Product (Element-wise multiplication):
[[ 2 12  8]
 [24 15 12]
 [30 63 24]]
```

6.Create an 8X3 integer array from a range between 10 to 34 such that the difference between each element is 2 and then Split the array into four equal-sized sub-arrays.

In [ ]:
```python
import numpy as np

array = np.arange(10, 34, 2).reshape(8, 3)
sub_arrays_rows = np.split(array, 2, axis=0)
sub_arrays = [np.split(sub_array, 2, axis=1) for sub_array in sub_arrays_rows]

print("Original 8x3 Array:")
print(array)
print("\nSub-Arrays:")
for i, row_sub_arrays in enumerate(sub_arrays):
    for j, sub_array in enumerate(row_sub_arrays):
        print(f"Sub-Array {i + 1}-{j + 1}:")
        print(sub_array)
```

7.Write a program to delete the second column from a given array and insert the new column in its place. Hint: Use delete function & insert function

Sample output

Original array

[[34 43 73][82 22 12][53 94 66]]

Array after deleting column 2 on axis 1 [[34 73][82 12][53 66]]

Array after inserting column 2 on axis 1 [[34 10 73][82 10 12][53 10 66]]

In [21]:
```
import numpy as np

original_array = np.array([[34, 43, 73],
                           [82, 22, 12],
                           [53, 94, 66]])
print("Original array:")
print(original_array)

modified_array = np.delete(original_array, 1, axis=1)

new_column = np.full((original_array.shape[0], 1), 10)

result_array = np.insert(modified_array, 1, new_column, axis=1)

print("\nArray after deleting column 2 on axis 1:")
print(modified_array)
print("\nArray after inserting column 2 on axis 1:")
print(result_array)
```

```
Original array:
[[34 43 73]
 [82 22 12]
 [53 94 66]]

Array after deleting column 2 on axis 1:
[[34 73]
 [82 12]
 [53 66]]

Array after inserting column 2 on axis 1:
[[34 10 10 10 73]
 [82 10 10 10 12]
 [53 10 10 10 66]]
```

8. Create a filter array that will return only values which are divisible by 3:

In [22]:
```
import numpy as np

array = np.array([10, 15, 18, 21, 25, 30, 36, 40])

filter_array = (array % 3 == 0)

result = array[filter_array]
print("Original Array:")
print(array)
print("\nFiltered Values Divisible by 3:")
print(result)
```

```
Original Array:
[10 15 18 21 25 30 36 40]

Filtered Values Divisible by 3:
[15 18 21 30 36]
```

9. Create a program that will play the “cows and bulls” game with the user. The game works like this:

Randomly generate a 4-digit number. Ask the user to guess a 4-digit number. For every digit that the user guessed correctly in the correct place, they have a “cow”. For every digit the user guessed correctly in the wrong place is a “bull.” Every time the user makes a guess, tell them how many “cows” and “bulls” they have. Once the user guesses the correct number, the game is over. Keep track of the number of guesses the user makes throughout the game and tell the user at the end.

Say the number generated by the computer is 1038. An example interaction could look like this:

Welcome to the Cows and Bulls Game!

Enter a number:1234

2 cows, 0 bulls

1856

1 cow, 1 bull

In [33]:
```
import random

def generate_random_number():
    return ''.join(random.sample('0123456789', 4))

def count_cows_bulls(random_number, user_guess):
    cows = sum(1 for r, g in zip(random_number, user_guess) if r == g)
    common_digits = set(random_number) & set(user_guess)
    bulls = len(common_digits) - cows
    return cows, bulls

def cows_and_bulls_game():
    random_number = generate_random_number()
    attempts = 0

    print("Welcome to the Cows and Bulls Game!")

    while True:
        user_guess = input("Enter a 4-digit number (or 'exit' to quit): ")

        if user_guess.lower() == 'exit':
            print(f"The random number was: {random_number}")
            print(f"You made {attempts} attempts.")
            break
```

```
        if len(user_guess) != 4 or not user_guess.isdigit():
            print("Please enter a valid 4-digit number.")
            continue

        attempts += 1
        cows, bulls = count_cows_bulls(random_number, user_guess)

        if cows == 4:
            print(f"Congratulations! You've guessed the correct number {random
_number} in {attempts} attempts.")
            break

        print(f"{cows} cows, {bulls} bulls")

if __name__ == "__main__":
    cows_and_bulls_game()
```

```
Welcome to the Cows and Bulls Game!
Enter a 4-digit number (or 'exit' to quit): 1234
0 cows, 1 bulls
Enter a 4-digit number (or 'exit' to quit): 2134
0 cows, 1 bulls
Enter a 4-digit number (or 'exit' to quit): 3124
0 cows, 1 bulls
Enter a 4-digit number (or 'exit' to quit): 2564
0 cows, 1 bulls
Enter a 4-digit number (or 'exit' to quit): 4567
0 cows, 0 bulls
Enter a 4-digit number (or 'exit' to quit): 1243
0 cows, 1 bulls
Enter a 4-digit number (or 'exit' to quit): 1342
1 cows, 0 bulls
Enter a 4-digit number (or 'exit' to quit): 6782
1 cows, 1 bulls
Enter a 4-digit number (or 'exit' to quit): 8762
2 cows, 0 bulls
Enter a 4-digit number (or 'exit' to quit): 8002
3 cows, 0 bulls
Enter a 4-digit number (or 'exit' to quit): 8702
2 cows, 1 bulls
Enter a 4-digit number (or 'exit' to quit): 8072
3 cows, 0 bulls
Enter a 4-digit number (or 'exit' to quit): 8092
Congratulations! You've guessed the correct number 8092 in 13 attempts.
```

10. Write a program to print the checkerboard pattern of n x n using NumPy.

Given n, print the checkerboard pattern for a n x n matrix considering that 0 for black and 1 for white.

In [25]:
```
import numpy as np

def create_checkerboard_pattern(n):
    checkerboard = np.zeros((n, n), dtype=int)

    checkerboard[1::2, ::2] = 1
    checkerboard[::2, 1::2] = 1

    return checkerboard

n = int(input("Enter the size of the checkerboard (n): "))

checkerboard = create_checkerboard_pattern(n)

print("Checkerboard Pattern (0 for black, 1 for white):")
print(checkerboard)
```

```
Enter the size of the checkerboard (n): 6
Checkerboard Pattern (0 for black, 1 for white):
[[0 1 0 1 0 1]
 [1 0 1 0 1 0]
 [0 1 0 1 0 1]
 [1 0 1 0 1 0]
 [0 1 0 1 0 1]
 [1 0 1 0 1 0]]
```

In [ ]: