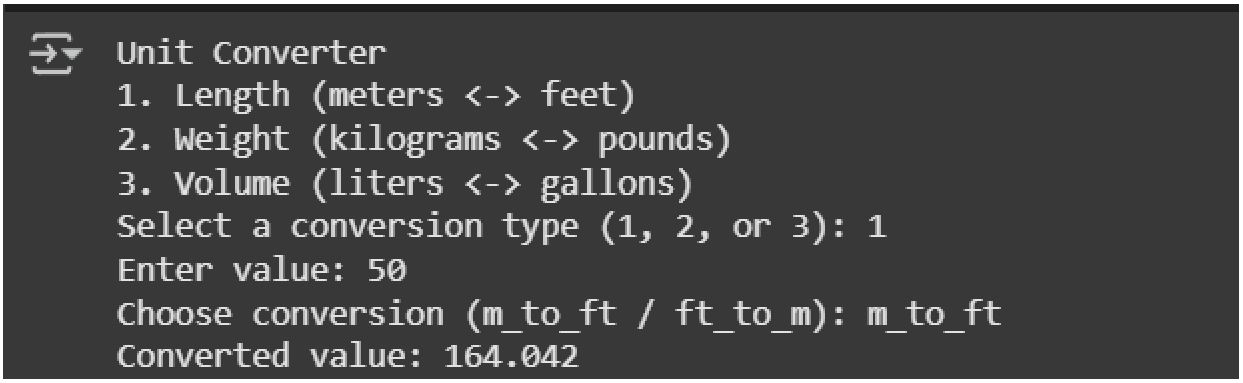
Krish Pradhan

Uni Id: 2359865

Exercise on Functions:

Task - 1:

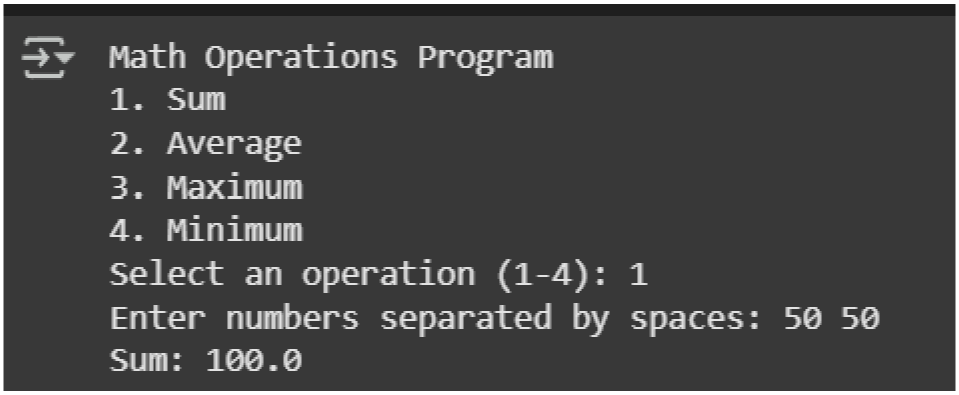
Create a Python program that converts between different units of measurement.



```
Unit Converter
1. Length (meters <-> feet)
2. Weight (kilograms <-> pounds)
3. Volume (liters <-> gallons)
Select a conversion type (1, 2, or 3): 1
Enter value: 50
Choose conversion (m_to_ft / ft_to_m): m_to_ft
Converted value: 164.042
```

Task - 2:

Create a Python program that performs various mathematical operations on a list of numbers.

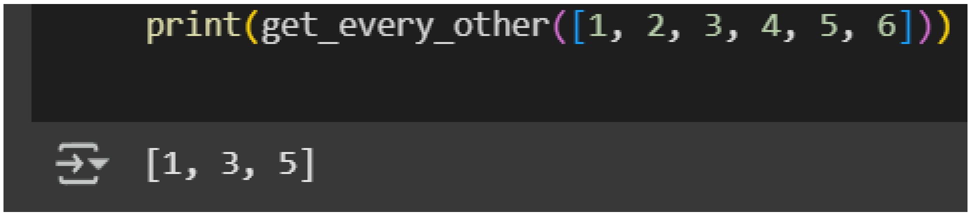


```
Math Operations Program
1. Sum
2. Average
3. Maximum
4. Minimum
Select an operation (1-4): 1
Enter numbers separated by spaces: 50 50
Sum: 100.0
```

4.2 Exercise on List Manipulation:

1. Extract Every Other Element:

Write a Python function that extracts every other element from a list, starting from the first element.

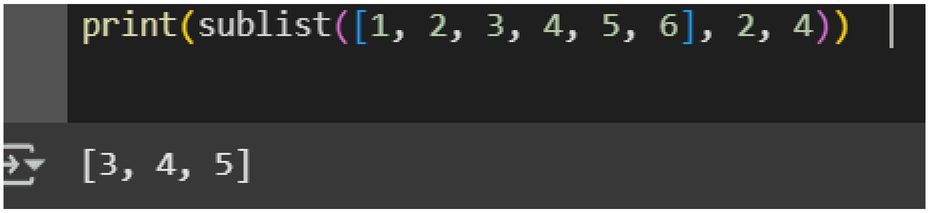


```
print(get_every_other([1, 2, 3, 4, 5, 6]))
```

```
[1, 3, 5]
```

2. Slice a Sublist:

Write a Python function that returns a sublist from a given list, starting from a specified index and ending at another specified index.

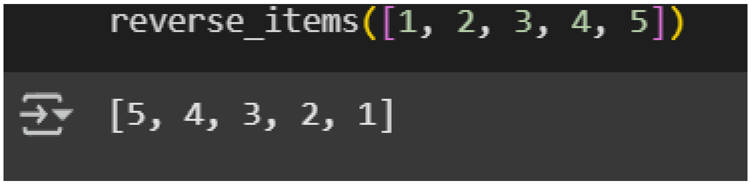


```
print(sublist([1, 2, 3, 4, 5, 6], 2, 4))
```

```
[3, 4, 5]
```

3. Reverse a List Using Slicing:

Write a Python function that reverses a list using slicing.

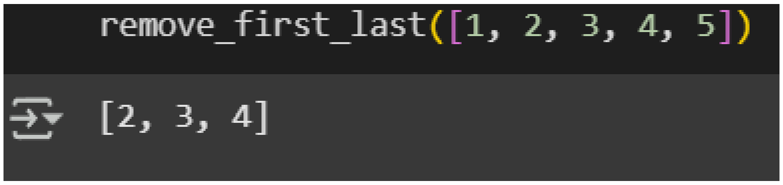


```
reverse_items([1, 2, 3, 4, 5])
```

```
[5, 4, 3, 2, 1]
```

4. Remove the First and Last Elements:

Write a Python function that removes the first and last elements of a list and returns the resulting sublist.

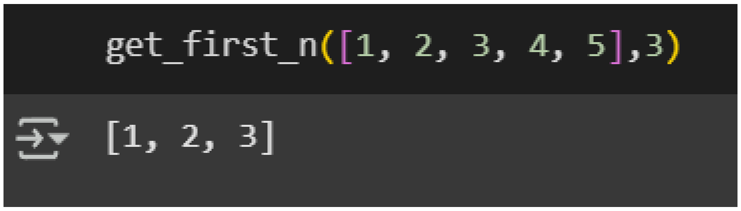


```
remove_first_last([1, 2, 3, 4, 5])
```

```
[2, 3, 4]
```

5. Get the First n Elements:

Write a Python function that extracts the first n elements from a list.

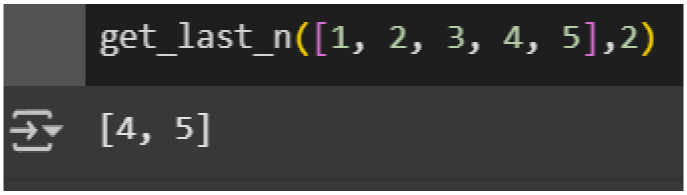


```
get_first_n([1, 2, 3, 4, 5],3)
```

```
[1, 2, 3]
```

6. Extract Elements from the End:

Write a Python function that extracts the last n elements of a list using slicing.

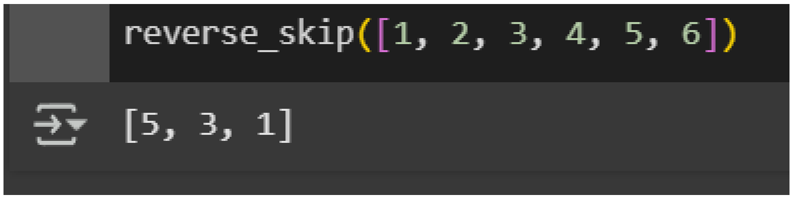


```
get_last_n([1, 2, 3, 4, 5],2)
```

```
[4, 5]
```

7. Extract Elements in Reverse Order:

Write a Python function that extracts a list of elements in reverse order starting from the second-to-last element and skipping one element in between.



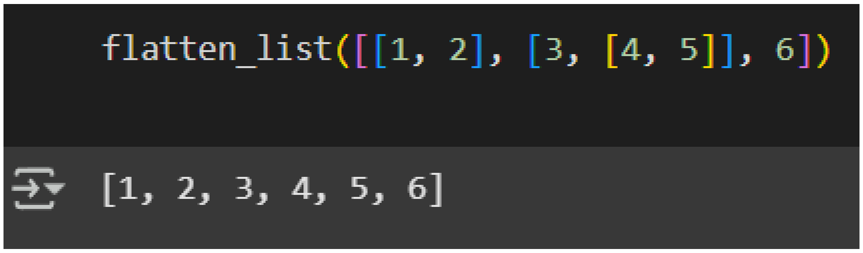
```
reverse_skip([1, 2, 3, 4, 5, 6])
```

```
[5, 3, 1]
```

4.3 Exercise on Nested List:

1. Flatten a Nested List:

Write a Python function that takes a nested list and flattens it into a single list, where all the elements are in a single dimension.

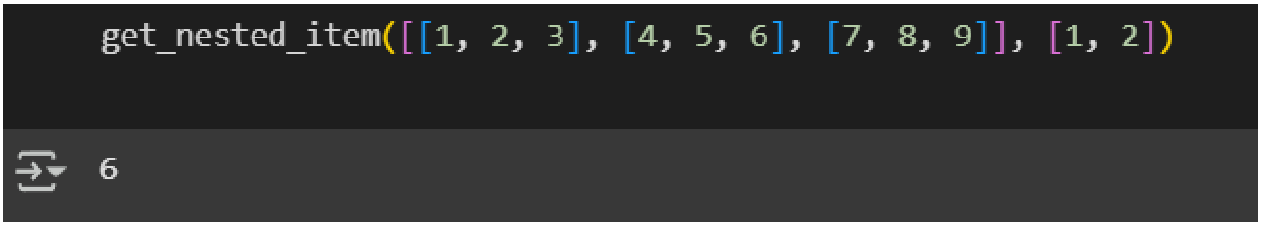


```
flatten_list([[1, 2], [3, [4, 5]], 6])
```

```
[1, 2, 3, 4, 5, 6]
```

2. Accessing Nested List Elements:

Write a Python function that extracts a specific element from a nested list given its indices.

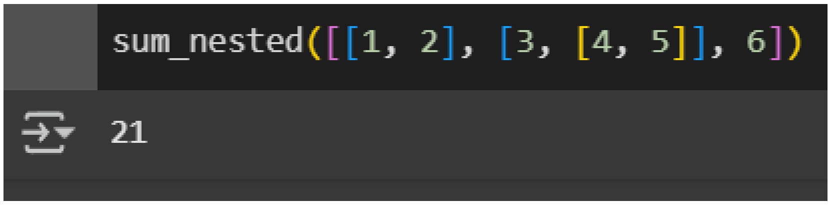


```
get_nested_item([[1, 2, 3], [4, 5, 6], [7, 8, 9]], [1, 2])
```

```
6
```

3. Sum of All Elements in a Nested List:

Write a Python function that calculates the sum of all the numbers in a nested list (regardless of depth).

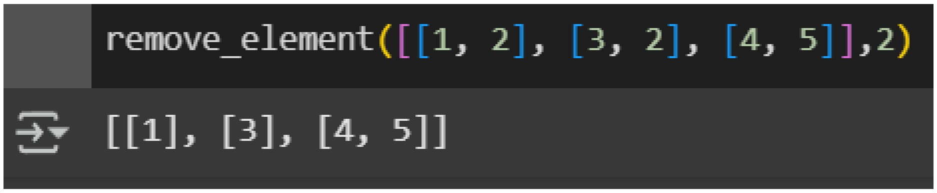


```
sum_nested([[1, 2], [3, [4, 5]], 6])
```

```
21
```

4. Remove Specific Element from a Nested List:

Write a Python function that removes all occurrences of a specific element from a nested list.

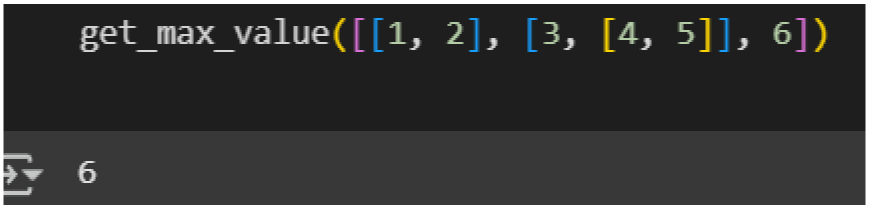


```
remove_element([[1, 2], [3, 2], [4, 5]],2)
```

```
[[1], [3], [4, 5]]
```

5. Find the Maximum Element in a Nested List:

Write a Python function that finds the maximum element in a nested list (regardless of depth).

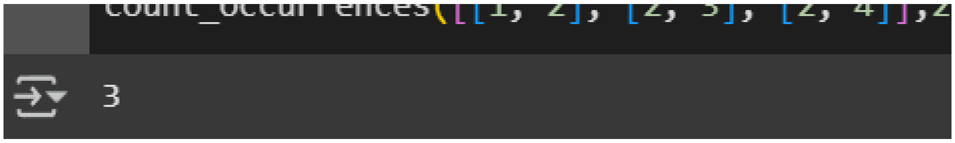


```
get_max_value([[1, 2], [3, [4, 5]], 6])
```

```
6
```

6. Count Occurrences of an Element in a Nested List:

Write a Python function that counts how many times a specific element appears in a nested list.

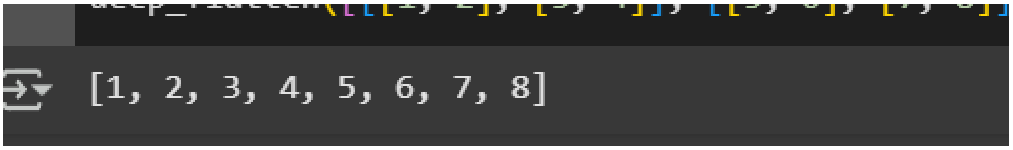


```
count_occurrences([[1, 2], [2, 3], [2, 4]],2
```

```
3
```

7. Flatten a List of Lists of Lists:

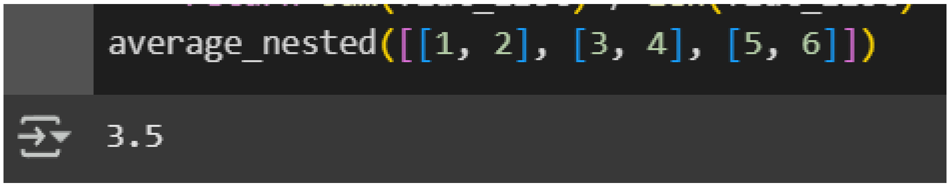
Write a Python function that flattens a list of lists of lists into a single list, regardless of the depth.



```
[1, 2, 3, 4, 5, 6, 7, 8]
```

8. Nested List Average:

Write a Python function that calculates the average of all elements in a nested list.

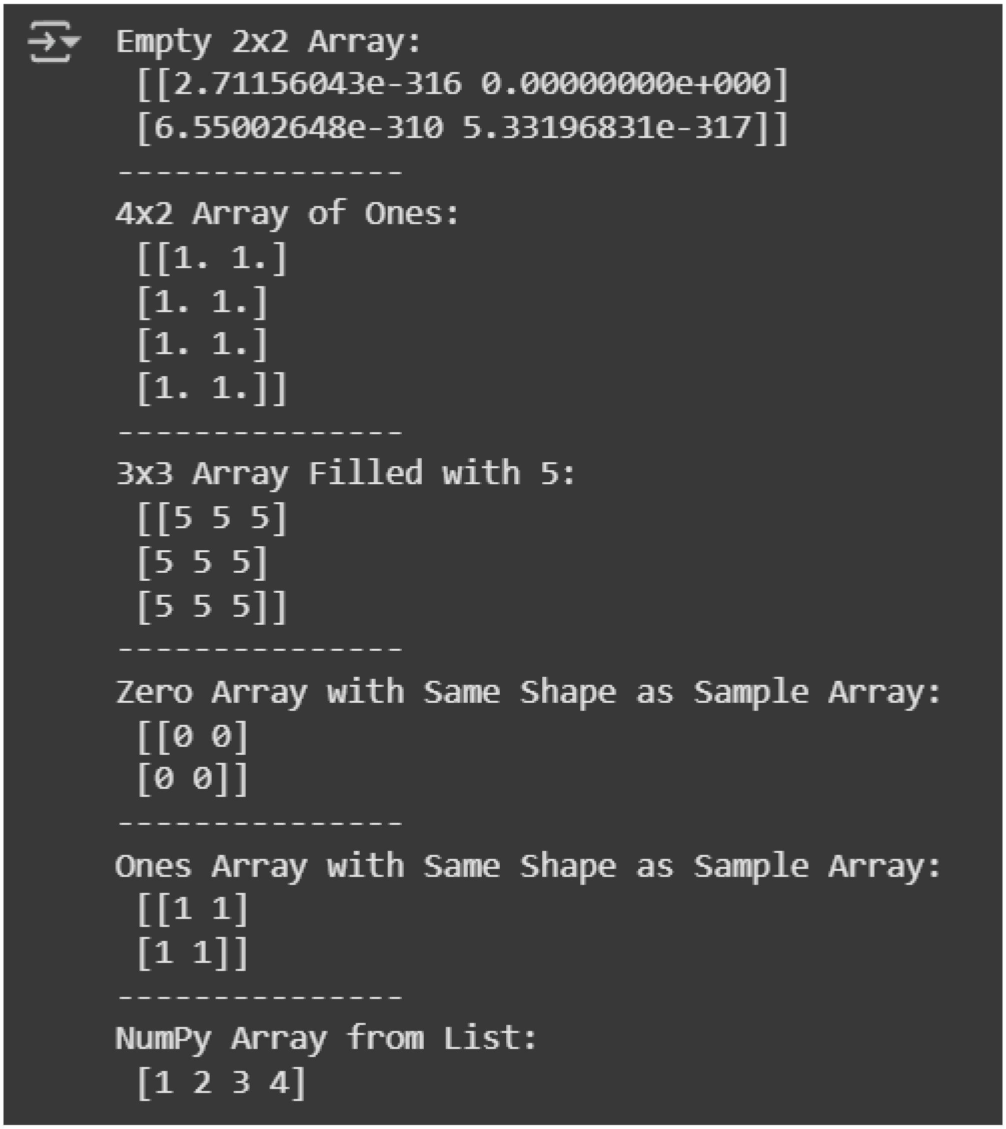


```
average_nested([[1, 2], [3, 4], [5, 6]])
```

```
3.5
```

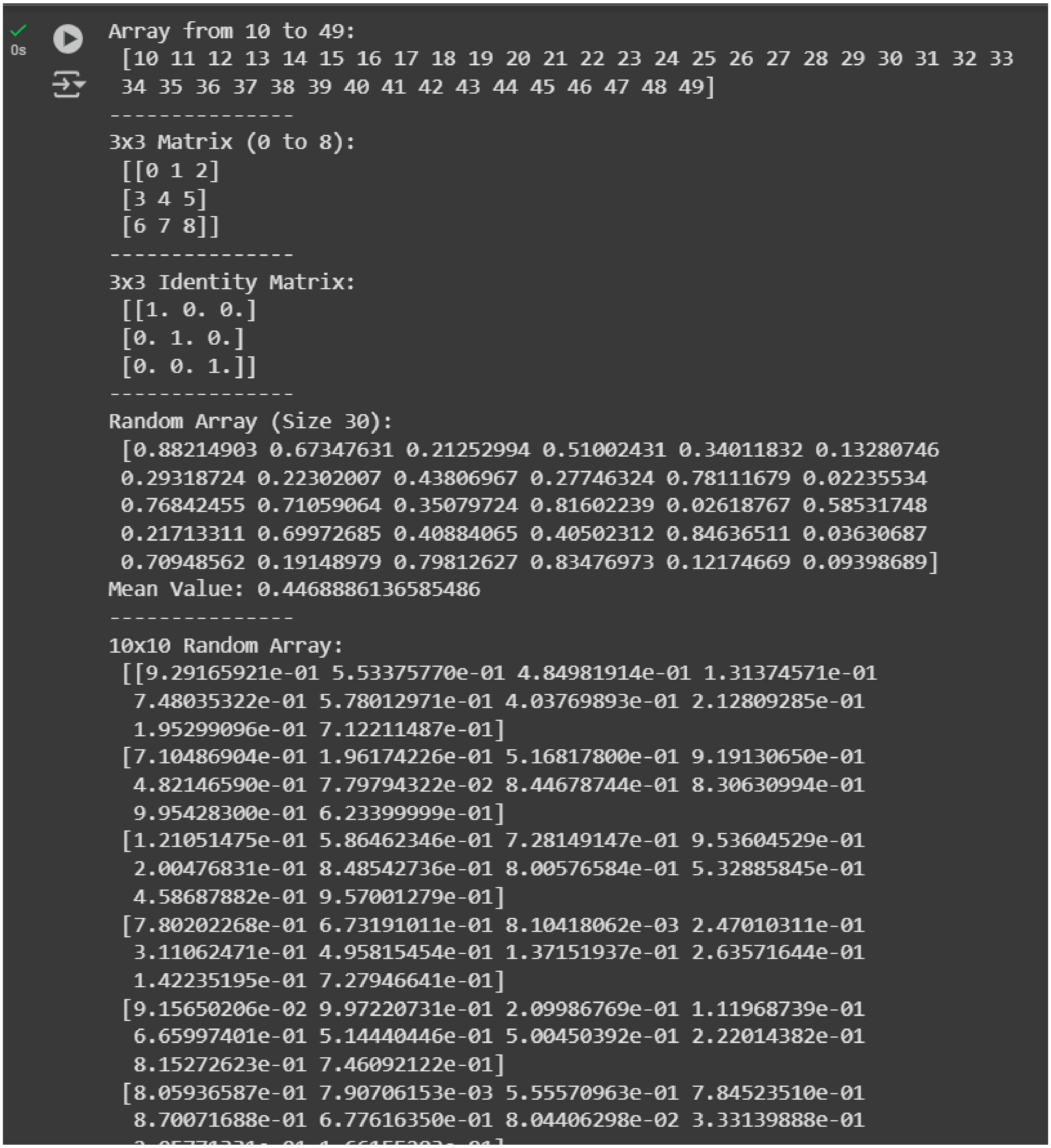
Basic Vector and Matrix Operation with Numpy.

Problem - 1: Array Creation:

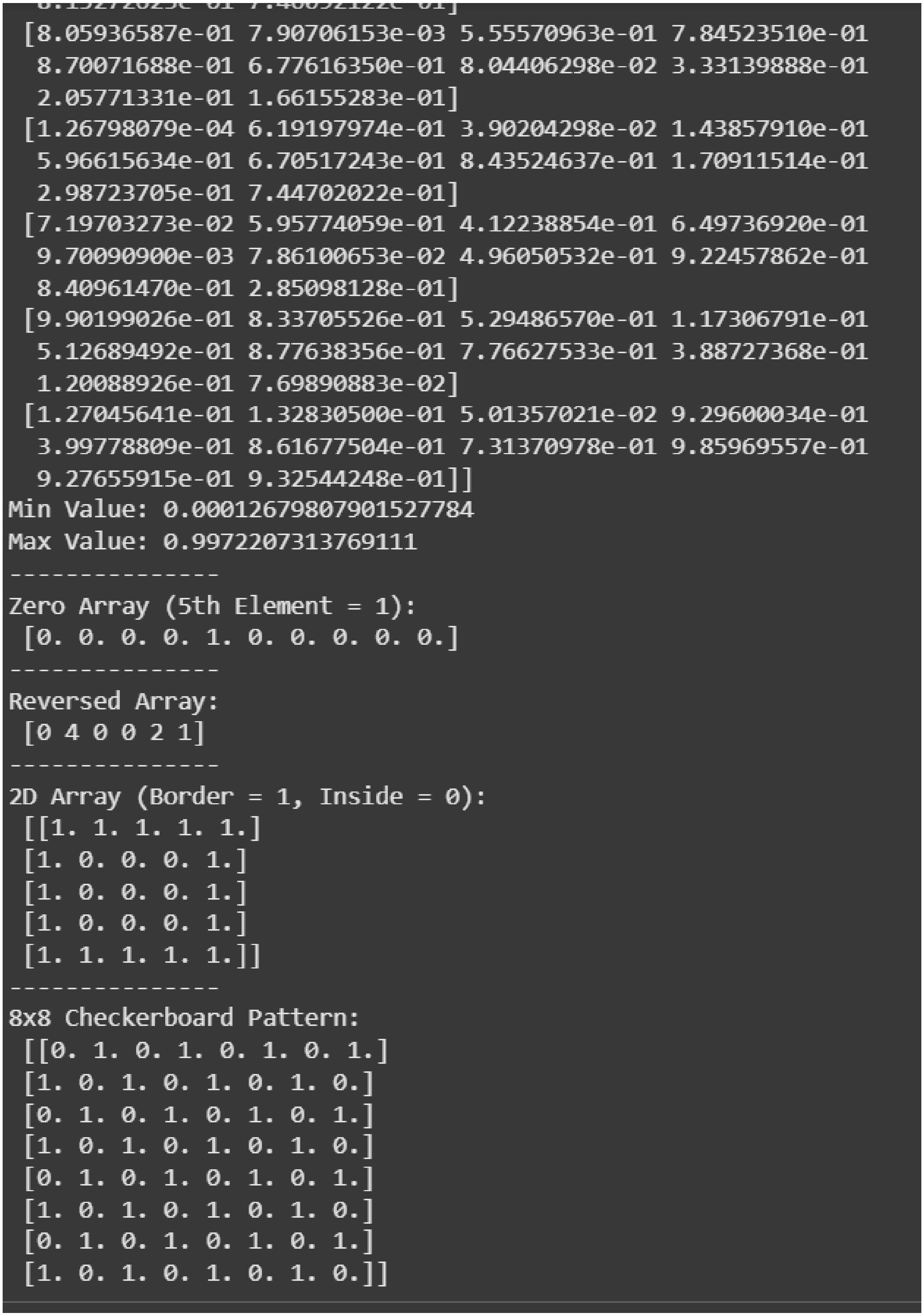


```
Empty 2x2 Array:
 [[2.71156043e-316 0.00000000e+000]
 [6.55002648e-310 5.33196831e-317]]
---------------
4x2 Array of Ones:
 [[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
---------------
3x3 Array Filled with 5:
 [[5 5 5]
 [5 5 5]
 [5 5 5]]
---------------
Zero Array with Same Shape as Sample Array:
 [[0 0]
 [0 0]]
---------------
Ones Array with Same Shape as Sample Array:
 [[1 1]
 [1 1]]
---------------
NumPy Array from List:
 [1 2 3 4]
```

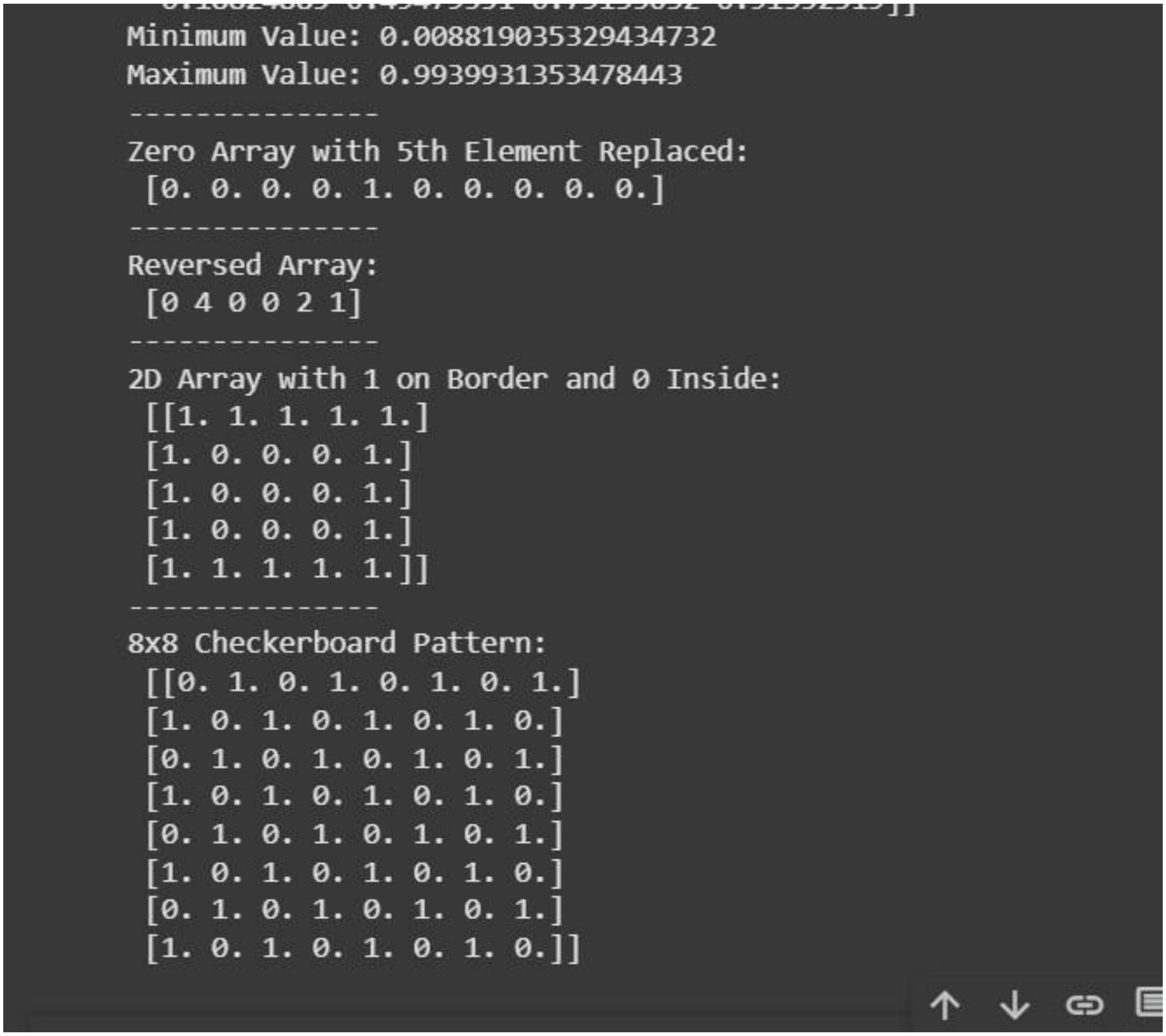
Problem - 2: Array Manipulation: Numerical Ranges and Array indexing:



```
Array from 10 to 49:
 [10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
---------------
3x3 Matrix (0 to 8):
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
---------------
3x3 Identity Matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
---------------
Random Array (Size 30):
 [0.88214903 0.67347631 0.21252994 0.51002431 0.34011832 0.13280746
 0.29318724 0.22302007 0.43806967 0.27746324 0.78111679 0.02235534
 0.76842455 0.71059064 0.35079724 0.81602239 0.02618767 0.58531748
 0.21713311 0.69972685 0.40884065 0.40502312 0.84636511 0.03630687
 0.70948562 0.19148979 0.79812627 0.83476973 0.12174669 0.09398689]
Mean Value: 0.4468886136585486
---------------
10x10 Random Array:
 [[9.29165921e-01 5.53375770e-01 4.84981914e-01 1.31374571e-01
  7.48035322e-01 5.78012971e-01 4.03769893e-01 2.12809285e-01
  1.95299096e-01 7.12211487e-01]
 [7.10486904e-01 1.96174226e-01 5.16817800e-01 9.19130650e-01
  4.82146590e-01 7.79794322e-02 8.44678744e-01 8.30630994e-01
  9.95428300e-01 6.23399999e-01]
 [1.21051475e-01 5.86462346e-01 7.28149147e-01 9.53604529e-01
  2.00476831e-01 8.48542736e-01 8.00576584e-01 5.32885845e-01
  4.58687882e-01 9.57001279e-01]
 [7.80202268e-01 6.73191011e-01 8.10418062e-03 2.47010311e-01
  3.11062471e-01 4.95815454e-01 1.37151937e-01 2.63571644e-01
  1.42235195e-01 7.27946641e-01]
 [9.15650206e-02 9.97220731e-01 2.09986769e-01 1.11968739e-01
  6.65997401e-01 5.14440446e-01 5.00450392e-01 2.22014382e-01
  8.15272623e-01 7.46092122e-01]
 [8.05936587e-01 7.90706153e-03 5.55570963e-01 7.84523510e-01
  8.70071688e-01 6.77616350e-01 8.04406298e-02 3.33139888e-01
  3.05771331e-01 1.66155383e-01]
```

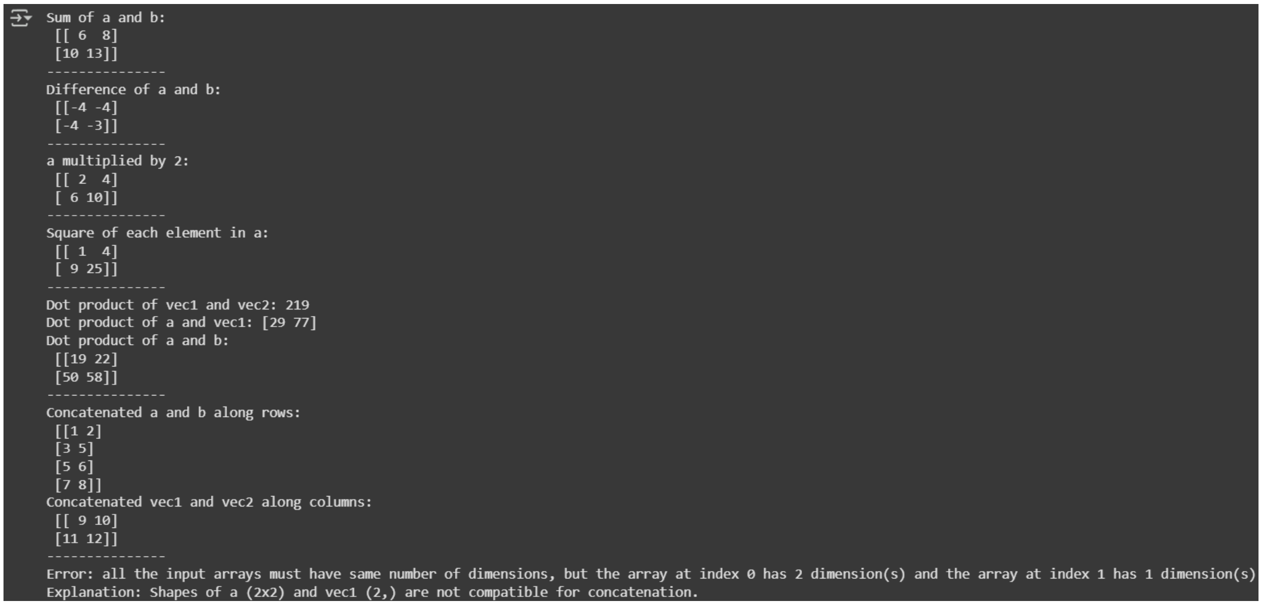


```
  8.15272825e-01 7.46692122e-01]
 [8.05936587e-01 7.90706153e-03 5.55570963e-01 7.84523510e-01
  8.70071688e-01 6.77616350e-01 8.04406298e-02 3.33139888e-01
  2.05771331e-01 1.66155283e-01]
 [1.26798079e-04 6.19197974e-01 3.90204298e-02 1.43857910e-01
  5.96615634e-01 6.70517243e-01 8.43524637e-01 1.70911514e-01
  2.98723705e-01 7.44702022e-01]
 [7.19703273e-02 5.95774059e-01 4.12238854e-01 6.49736920e-01
  9.70090900e-03 7.86100653e-02 4.96050532e-01 9.22457862e-01
  8.40961470e-01 2.85098128e-01]
 [9.90199026e-01 8.33705526e-01 5.29486570e-01 1.17306791e-01
  5.12689492e-01 8.77638356e-01 7.76627533e-01 3.88727368e-01
  1.20088926e-01 7.69890883e-02]
 [1.27045641e-01 1.32830500e-01 5.01357021e-02 9.29600034e-01
  3.99778809e-01 8.61677504e-01 7.31370978e-01 9.85969557e-01
  9.27655915e-01 9.32544248e-01]]
Min Value: 0.00012679807901527784
Max Value: 0.9972207313769111
---------------
Zero Array (5th Element = 1):
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
---------------
Reversed Array:
 [0 4 0 0 2 1]
---------------
2D Array (Border = 1, Inside = 0):
 [[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
---------------
8x8 Checkerboard Pattern:
 [[0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]]
```



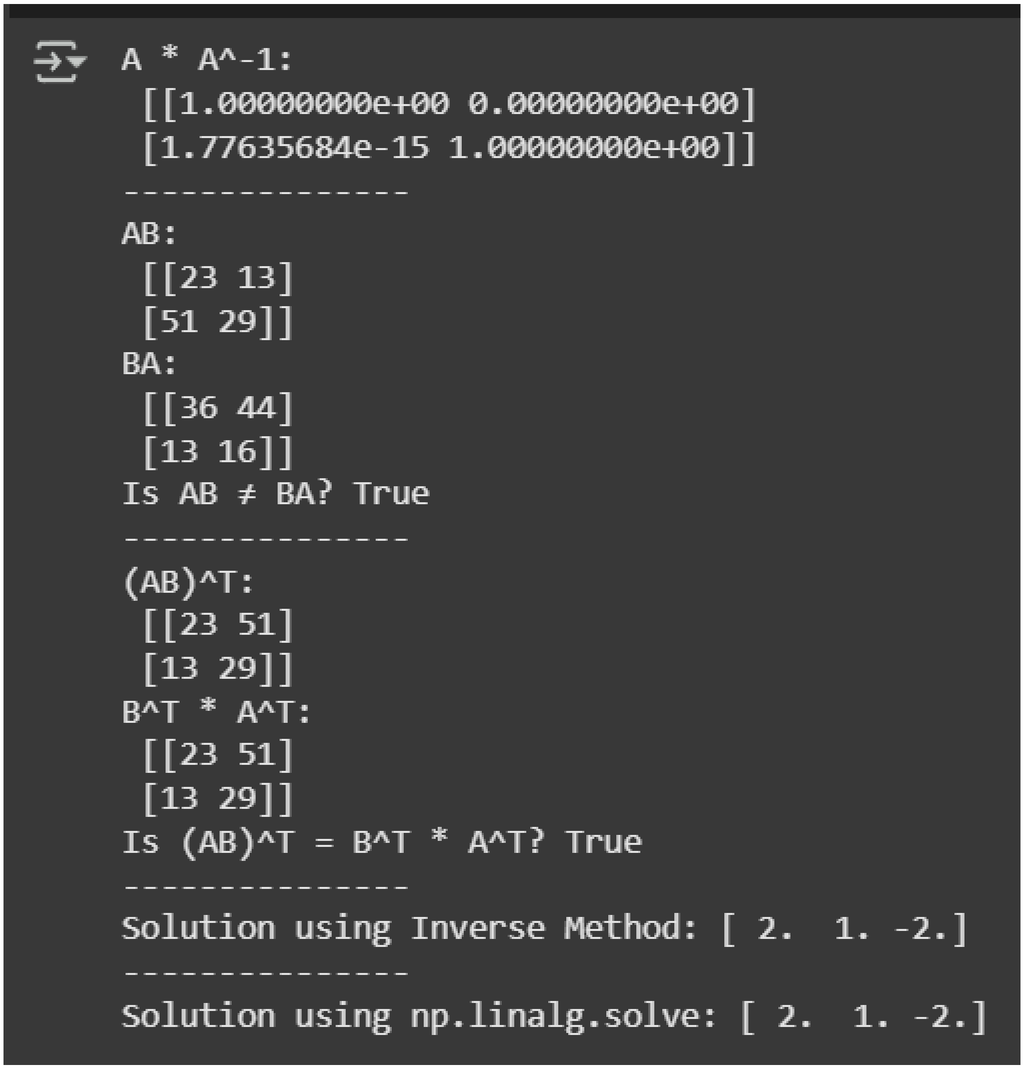
```
Minimum Value: 0.008819035329434732
Maximum Value: 0.9939931353478443
---------------
Zero Array with 5th Element Replaced:
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
---------------
Reversed Array:
 [0 4 0 0 2 1]
---------------
2D Array with 1 on Border and 0 Inside:
 [[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
---------------
8x8 Checkerboard Pattern:
 [[0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]]
```

Problem - 3: Array Operations:



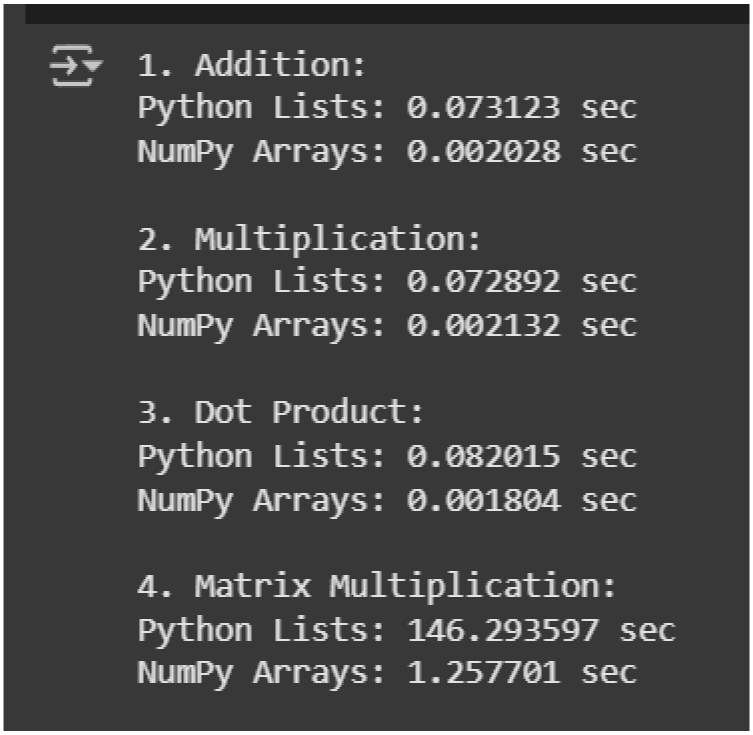
```
Sum of a and b:
 [[ 6  8]
 [10 13]]
---------------
Difference of a and b:
 [[-4 -4]
 [-4 -3]]
---------------
a multiplied by 2:
 [[ 2  4]
 [ 6 10]]
---------------
Square of each element in a:
 [[ 1  4]
 [ 9 25]]
---------------
Dot product of vec1 and vec2: 219
Dot product of a and vec1: [29 77]
Dot product of a and b:
 [[19 22]
 [50 58]]
---------------
Concatenated a and b along rows:
 [[1 2]
 [3 5]
 [5 6]
 [7 8]]
Concatenated vec1 and vec2 along columns:
 [[ 9 10]
 [11 12]]
---------------
Error: all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimension(s) and the array at index 1 has 1 dimension(s)
Explanation: Shapes of a (2x2) and vec1 (2,) are not compatible for concatenation.
```

Problem - 4: Matrix Operations:



```
A * A^-1:
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]
---------------
AB:
 [[23 13]
 [51 29]]
BA:
 [[36 44]
 [13 16]]
Is AB ≠ BA? True
---------------
(AB)^T:
 [[23 51]
 [13 29]]
B^T * A^T:
 [[23 51]
 [13 29]]
Is (AB)^T = B^T * A^T? True
---------------
Solution using Inverse Method: [ 2.  1. -2.]
---------------
Solution using np.linalg.solve: [ 2.  1. -2.]
```

Experiment: How Fast is Numpy?



```
1. Addition:
Python Lists: 0.073123 sec
NumPy Arrays: 0.002028 sec

2. Multiplication:
Python Lists: 0.072892 sec
NumPy Arrays: 0.002132 sec

3. Dot Product:
Python Lists: 0.082015 sec
NumPy Arrays: 0.001804 sec

4. Matrix Multiplication:
Python Lists: 146.293597 sec
NumPy Arrays: 1.257701 sec
```